# Contents

## THEORY OF COMPUTATION (V-SEM., IT-BRANCH)



# UNIT 1

## INTRODUCTION OF THE THEORY OF COMPUTATION, FINITE STATE AUTOMATA – DESCRIPTION OF FINITE AUTOMATA, PROPERTIES OF TRANSITION FUNCTIONS, TRANSITION GRAPH, DESIGNING FINITE AUTOMATA, FSM, DFA, NFA, 2-WAY FINITE AUTOMATA, EQUIVALENCE OF NFA AND DFA, MEALY AND MOORE MACHINES

***Q.1. What do you mean by theory of computation ?***

***Ans.*** Theory of computation is a branch of computer science that deals with how efficiently a problem can be solved on a model of computation using algorithm.

***Q.2. Write short note on automata.*** ***(R.G.P.V., June 2010)***

***Ans.*** An ***automaton*** is an abstract model of a digital computer. Every automaton includes some essential features. It has a mechanism for reading input. It is assumed that the input is a string over a given alphabet, written on a file known as ***input tape***, which the automaton can only read. This file is divided into cells, each of which can hold one symbol. The input mechanism can scan (or read) the file from left to right, one symbol at a time. Also the input mechanism can detect the end of the input string (sensing end-of-file condition). The output of some form can be produced by automaton. The automaton may have a temporary ***storage*** device, having an unlimited number of cells, each of which can hold a single symbol from an alphabet (not necessarily the same as the input alphabet). The automaton can read as well as change the contents of the storage cells. Finally, it has a ***control unit***, which can be in any one of a

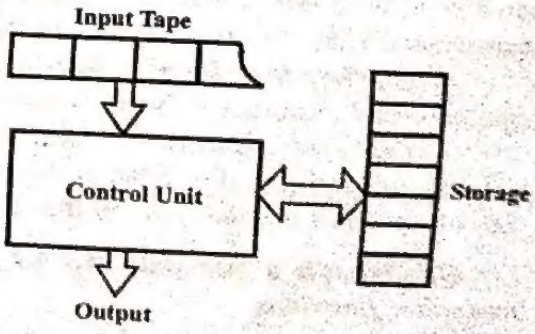



***Fig. 1.1 A General Automaton***

finite number of ***internal states.*** The control unit can change state in some defined manner. Fig. 1.1 shows a diagram of a general automaton.

An automaton is supposed to operate in a discrete time frame. At some given time, the control unit is in some internal state, and the input mechanism is scanning a specific symbol on the input tape. The internal state of the control unit at the next step is obtained by the ***next-state*** or ***transition function.*** This transition function gives the next state in terms of the current state, the current input symbol on the input tape, and the information currently in the temporary storage. During the transition from one time interval to the next, output may be generated or the information in the temporary storage changed. The term ***configuration*** refers to a particular state of the control unit, input tape, and temporary storage. The transition from one configuration to the next is called a ***move.***

***Q.3. Write short note on finite automata.***

***Ans.*** Finite automata is denoted by 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

(i) $Q$ : finite set of states.
(ii) $\Sigma$: finite set of inputs.
(iii) $\delta$: transition function which maps $Q \times \Sigma \rightarrow Q$.
(iv) $q_0$ : initial state, $q_0 \in Q$.
(v) $F$ : set of final states such that $F \subseteq Q$.

The above model is represented graphically in fig. 1.2.

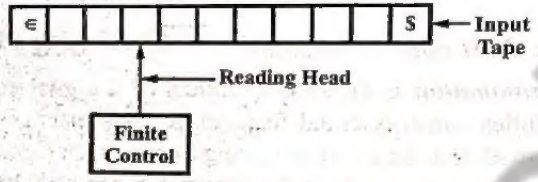



***Fig. 1.2 Block Diagram of a Finite Automaton***

Various components of a finite automaton are –

***(i) Input Tape*** – Input tape stores input string. It is divided into squares, each square containing a single symbol from the input string. String is processed from left to right in input tape.

***(ii) Reading Head*** – The head moves from square to square in input tape and examines only one square at a time.

***(iii) Finite Control*** – The finite control takes the input which is currently under the reading head or the present state of the machine and gives the output which is calculated by transition function on the inputs.

***Q.4. What is a trap state in FA ? State and explain the properties of transition functions.*** ***(R.G.P.V., Dec. 2015)***

***Ans.*** The automata in fig. 1.3 remains in its initial state $q_0$ until the first $b$ is encountered. If this is also the last symbol of the input, then the string is accepted.



If not, the DFA goes into state $q_2$, from which it can never escape. Such a state is called a trap state.

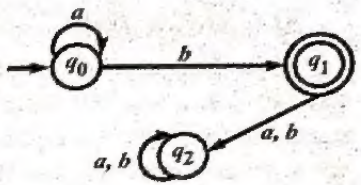



***Fig. 1.3***

The properties of transition functions are as follows –

(i) $\delta(q, \Lambda) = q$ in a finite automaton. This means the state of the system can be changed only by an input symbol.

(ii) For all strings $w$ and input symbols $a$,

$$\delta(q, aw) = \delta(\delta(q, a), w)$$
$$\delta(q, wa) = \delta(\delta(q, w), a)$$

This property gives the state after the automaton consumes or reads the first symbol of a string $aw$ and the state after the automaton consumes a prefix of the string $wa$.

***Q.5. What is the use of transition graph ? What are they ?***

***Ans.*** A directed graph, called a ***transition graph*** or transition diagram, is used to visualize and represent finite automata. It is just a simple graph with vertices representing states and edges representing transitions. The vertices are represented by circles and the edges are represented by lines with arrows connecting the vertices. The labels on the vertices are the names of the states, while the labels on the edges are the current input symbol values. For example, if $q_0$ and $q_1$ are internal states of a DFA $M$, then its associated graph will have two vertices labelled $q_0$ and $q_1$. An edge $(q_0, q_1)$ labelled $a$ represents the transition $\delta(q_0, a) = q_1$. The initial state is represented by an incoming unlabelled arrow not originating at any vertex. One or more, final states are shown by double circles.

Formally, we can say, if $M = (Q, \Sigma, \delta, q_0, F)$ is a deterministic finite accepter, its associated transition graph $G_M$ has exactly $|Q|$ vertices, each of which labelled with a different $q_i \in Q$. For each and every transition rule $\delta(q_i, a) = q_j$, the graph has an edge $(q_i, q_j)$ labelled $a$. The vertex with label $q_0$ is called the ***initial vertex,*** while the vertices labelled with $q_f \in F$ are the ***final vertices.*** It is a trivial matter to convert from the $(Q, \Sigma, \delta, q_0, F)$ definition of a DFA to its corresponding transition graph and vice-versa. Fig. 1.4 represents the DFA,

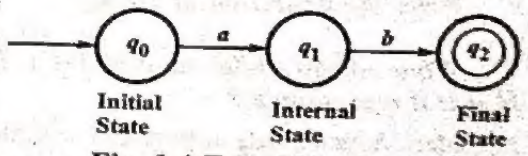



***Fig. 1.4 Transition Graph***

$$M = (\{q_0, q_1, q_2\}, \{a,b\}, \delta, \{q_2\})$$

where $\delta$ is given by

$$\delta(q_0, a) = q_1$$
$$\delta(q_1, b) = q_2$$

***Q.6. Define finite state machine (FSM) with example.***

***Ans.*** This machine will consist of input-output relation at every state and also the changes of the states that will occur on receiving the input at a particular state. Hence we will require the mapping in two forms. At a particular state, for a given input, what is the output.

$$S \times I \rightarrow O$$

and at a particular state on receiving the input, what is the next state.

$$S \times I \rightarrow S$$

These are respectively known as machine function (M.A.F.) and state function (S.T.F.). Both of them are functions of two arguments, current state, and current input but their results are different. The machine can be defined as,

(i) A finite set of states

$$S = \{S_0, S_1, S_2, .....\}$$

(ii) A special element of set '$S$' so called as initial state or start state

(iii) An input alphabet, $I = \{i_0, i_1, .....\}$

(iv) An output alphabet, $O = \{O_0, O_1, .....\}$

(v) A function $S \times I \rightarrow S$ (S.T.F.)

(vi) A function $S \times I \rightarrow O$ (M.A.F.).

At any state the machine is in any of it's states and on arrival of an input symbol it will change the state using state function and will also give the output using machine function. As the number of states is finite, it is known as finite state machine. The class of machines is known as finite automaton.

***Q.7. Give a detailed explanation of a deterministic finite acceptor.***

***Or***

***Discuss the finite state system with the help of a appropriate example.*** ***(R.G.P.V., June 2003)***

***Or***

***Write a short note on one way finite automata.*** ***(R.G.P.V., Dec. 2002)***

***Or***

***Write the definition of DFA.*** ***(R.G.P.V., Dec. 2014)***

***Or***

***What do you understand by DFA (Deterministic Finite Automata) and how is it represented ?*** ***(R.G.P.V., June 2015)***

***Ans.*** A finite state automaton or finite automaton deterministic finite state machine (DFSM) or deterministic finite accepter (DFA) is a 5 tuple machine that can be represented as,

$$M = (Q, \Sigma, \delta, q_0, F)$$

where, (i) $Q$ is a finite non-empty set of states

(ii) $\Sigma$ is a finite non-empty set of input symbols called terminals or input alphabets



(iii) $\delta$ is a function which maps $Q \times \Sigma$ into $Q$ and is usually called direct transition function. This function describes the change of states during the transition. Usually, this mapping is represented by a transition table or a transition diagram

(iv) $q_0$ is the initial state and $q_0 \in Q$

(v) $F$ is the set of final state and $F \subseteq Q$.

For example, the operation of a deterministic finite accepter is as follows –

Initially, it is assumed to be in the initial state $q_0$, with its input mechanism on the leftmost symbol of the input string. During every move of the automaton, the input mechanism shifts one position to the right, so each move consumes one input symbol. At the end of the string (i.e., after having all input symbols been read), it is accepted if the automaton is in one of its final states, otherwise it is rejected. The input mechanism can only move from left to right, it cannot move in reverse direction. Also, it can read only one input symbol at a time (i.e., on each step). The transition function $\delta$ governs the transition from one internal state to another. For example, if $\delta(q_0, a) = q_1$, then, if initially the DFA is in state $q_0$ and the current input symbol is $a$, the DFA will go into state $q_1$.

***Q.8. Write a short note on non-deterministic finite automata.*** ***(R.G.P.V., Dec. 2002)***

***Ans.*** Non-determinism means a choice of moves for an automaton. Instead of prescribing a unique move in every situation, this allows a set of possible moves. Formally, this can be achieved by defining the transition function such that its range is a set of possible states.

A non-deterministic finite accepter or NDFA is defined as a 5-tuple machine –

$$M = (Q, \Sigma, \delta, q_0, F)$$

where, (i) $Q$ is a finite nonempty set of states

(ii) $\Sigma$ is a finite nonempty set of input symbols

(iii) $\delta$ is the transition function mapping from $Q \times \Sigma$ into $2^Q$ which is the power set of $Q$, i.e., the set of all subsets of $Q$

(iv) $q_0$ is the initial state and $q_0 \in Q$

(v) $F$ is the set of final states and $F \subseteq Q$.

The transition function $\delta$ is also defined as,

$$\delta: Q \times (\Sigma \cup \{\Lambda\}) \rightarrow 2^Q$$

There are three major differences between DFA and NDFA. In an NDFA, the range of $\delta$ is in the power set $2^Q$, so that its value is not a single symbol of $Q$, but a subset of it, which defines the set of all possible states that can be reached by the transition. For example, if the current state is $q_1$, the symbol read is $a$, and

$\delta(q_1, a) = \{q_0, q_1\}$

then either $q_0$ or $q_2$ could be the next state of the NDFA. Also, we allow $\Lambda$ as the second argument of $\delta$. It means that the NDFA can make a transition without consuming an input symbol. Although in NDFA also, the input mechanism can only travel to the right it may become stationary on some moves. Moreover, in an NDFA, the transition $\delta(q_i, a)$ may be empty, i.e., there is no transition defined for this particular situation.

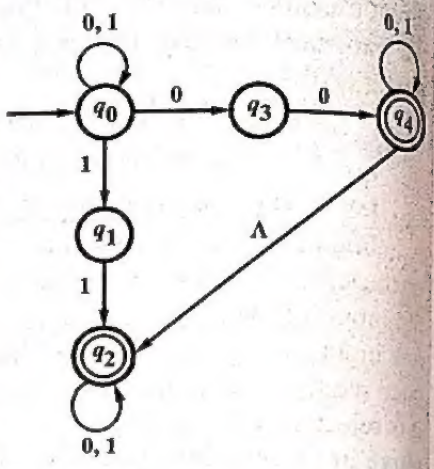



*Fig. 1.5 The Transition Diagram for an NDFA*

Like DFAs, NDFAs can be represented by transition graphs. The vertices can be determined by $Q$, while an edge $(q_i, q_j)$ with label $b$ is in the graph if and only if $\delta(q_i, b)$ contains $q_j$. But, since $b$ may be the empty string, so there can be some edges with labelled $\Lambda$ or $\in$ or $\lambda$.

For example, consider the transition diagram for an NDFA given in fig. 1.5.

The sequence of states for the input string 0100 is shown in fig. 1.6.

An input string is accepted by an NDFA, if there is some sequence of possible moves that put the machine in a final state at the end of the string (i.e., after having the last symbol been read), and is rejected only when there is no possible sequence of moves by which any final state can be reached.

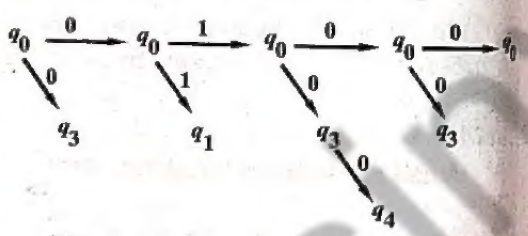



*Fig. 1.6 Sequence of States while Processing 0100*

In the above example shown in fig.1.5 and fig. 1.6, since $q_4$ is the final state, the input string 0100 will be accepted by the NDFA through the transition $\delta(q_0, 0100) = \{q_0, q_3, q_4\}$.

***Q.9. Define two-way finite automata with suitable example.***

***(R.G.P.V., Nov. 2018)***

***Or***

***Define two-way deterministic finite automaton (2DFA).***

***Or***

***Write short note on two-way finite automata.***

***(R.G.P.V., Dec. 2002, June 2010)***

***Or***

***Define two-way finite automata.***

***(R.G.P.V., Dec. 2016)***



***Ans.*** The finite automaton is a control unit that reads a tape, moving one square right at each move. We added non-determinism to the model, which allow many "copies" of the control unit to exist and scan the tape simultaneously. Next we added $\Lambda$-transitions, which allowed change of state without reading the input symbol or moving the tape head. Another extension is to allow the tape head the ability to move left as well as right. Such a finite automaton is called a ***two-way finite automaton or 2-FA.*** It accepts an input string if it moves the tape head off the right end of the tape, at the same time entering an accepting state. We shall see that even this generalization does not increase the power of the finite automaton; two-way FA accepts only regular sets. We give a proof only for a special case of a two-way FA, that is deterministic and whose tape head must move left or right, i.e., not stationary, at each move.

A two-way deterministic finite automaton or 2DFA is a quintuple

$$M = (Q, \Sigma, \delta, q_0, F)$$

where, $Q$ = A finite set of internal state

$\Sigma$ = A finite set of symbols called the input alphabets

$q_0$ = The initial state

$F$ = A set of final state as before and

$\delta$ = Transition function which maps from $Q \times \Sigma$ to $Q \times \{L, R\}$.

If $\delta(q, a) = (p, L)$ then in state $q$, scanning input symbol $a$, the 2DFA enters state $p$ and moves its head left one square. If $\delta(q, a) = (p, R)$, the 2DFA enters state $p$ and moves its head right one square.

**Example** – Consider a 2 DFA 'M' that behaves as follows, starting in state '$q_0$', '$M$' repeat a cycle of moves where in the tape head moves right until two 1's have been encountered, the left until encountering a '0', at which point state '$q_0$', is reentered and the cycle repeated '$M$' has three states, all of which are final, '$\delta$' is given in the table

| $Q$ \ $\Sigma$ | 0 | 1 |
|---|---|---|
| $q_0$ | $q_0, R$ | $q_1, R$ |
| $q_1$ | $q_1, R$ | $q_2, L$ |
| $q_2$ | $q_0, R$ | $q_2, L$ |

and input string is 101001.

Since $q_0$ is the initial state, the first ID is

$q_0101001$

To obtain the second 'ID', note that the symbol to the immediate 'Right' of the state '$q_0$' in the first 'ID' is a '1' and $\delta(q_0, 1)$ is $(q_1, R)$ thus the second

ID is $\boxed{q_1,01001}$ continue in this way we get the result.

$q_0\,101001$

$\xrightarrow{q_1,1,R}$

$\vdash 1q_1\,0\,1001$

$\xrightarrow{q_1,R}$

$\vdash 10\,q_1\,1001$ move till we get one 'zero'

$\xrightarrow{q_2,L}$

$\vdash 1q_2\,01001$

$\xrightarrow{q_0,R}$

$\vdash 10q_0\,1001$

$\xrightarrow{q_1,R}$

$\vdash 101q_1\,001$

$\xrightarrow{q_1,R}$

$\vdash 1010q_1\,01$

$\xrightarrow{q_1,R}$

$\vdash 10100q_1\,1$

$\xleftarrow{q_2,L}$

$\vdash 1010q_2\,01$

$\xrightarrow{q_0,R}$

$\vdash 10100q_0\,1$

$\xrightarrow{q_1,R}$

$\vdash 101001q_1$

***Q.10. Write short note on equivalent of DFA and NDFA.***

***(R.G.P.V., Dec. 2002, June 2010)***

***Ans.*** Since every DFA is an NDFA, it is clear that the class of languages accepted by NDFAs includes the regular sets (the language accepted by DFAs). However, it turns out that these are the only sets accepted by NDFAs. For every NDFA, we can construct an equivalent DFA that accepts the same language. The way a DFA simulates an NDFA is to allow the states of DFA to correspond to sets of states of the NDFA. The constructed DFA keeps track



in its finite control of all states that the NDFA could be in after reading the same input as the DFA has read.

Two finite accepters $M_1$ and $M_2$ are said to be equivalent if

$$L(M_1) = L(M_2)$$

i.e., if they both accept the same languages.

Let us take an example of the DFA shown in fig. 1.7. It is equivalent to the NDFA as shown in fig. 1.8. Because they both accept the same language.

$L(M) = \{w/w$ ends with $ab\}$

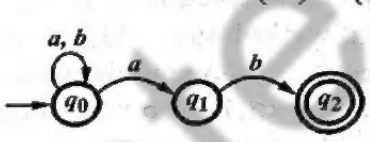



*Fig. 1.7*

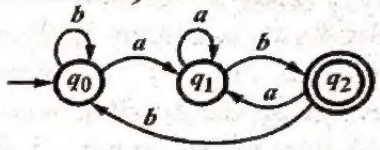



*Fig. 1.8*

***Q.11. Write a procedure that converts an NDFA to DFA.***

***Ans.*** An NDFA can be converted into DFA by following steps –

(i) Create a graph $G_D$ having vertex $\{q_0\}$. Identify this vertex as the initial one.

(ii) Repeat the following steps (a to e) until no more edges are missing –

(a) Take any vertex $\{q_i, q_j, \ldots\ldots, q_k\}$ of graph $G_D$ that has no outgoing edge for some $a \in \Sigma$.

(b) Compute $\delta^*(q_i,a), \delta^*(q_j,a), \ldots\ldots\ldots, \delta^*(q_k,a)$.

(c) From the union of all these $\delta^*$, yielding the set $\{q_l, q_m, \ldots\ldots\ldots, q_n\}$.

(d) Create a vertex for graph $G_D$ labelled $\{q_l, q_m, \ldots\ldots\ldots, q_n\}$ if it does not already present.

(e) Add an edge to graph $G_D$ from $\{q_i, q_j, \ldots\ldots\ldots, q_k\}$ to $\{q_l, q_m, \ldots\ldots\ldots, q_n\}$ and label it by $a$.

(iii) Every state of the graph $G_D$ whose label contains any $q_f \in F_N$ is defined as a final vertex.

(iv) If $M_N$ accepts $\Lambda$, the vertex $\{q_0\}$ in graph $G_D$ is also made a final vertex.

The procedure also terminates. Each pass through the loop in step (ii) adds an edge to graph $G_D$. But $G_D$ has at most $2^{|Q_N|}\,|\Sigma|$ edges, so that the loop eventually stops.

***Q.12. Prove that for every NDFA, there exists a DFA which simulates the behaviour of NDFA.***

***Or***

***Prove that if L is the set accepted by NDFA, then there exists a DFA which also accepts L.***

***Ans.*** **Proof** – Let $M_N = (Q, \Sigma, \delta, q_0, F)$ be an NDFA accepting $L$. We construct a DFA, $M_D$, as follows –

$$M_D = (Q', \Sigma, \delta', q_0', F')$$

where, (i) $Q' = 2^Q$ (any state in $Q'$ is denoted by $[q_1, q_2, q_3, \ldots\ldots q_j]$ where $q_1, q_2, q_3, \ldots\ldots q_j \in Q$

(ii) $q_0' = [q_0]$

(iii) $F'$ is the set of all subsets of $Q$ containing an element of $F$.

Before going to define $\delta'$, we look at the construction of $Q'$, $q_0'$, and $F'$. $M_N$ is initially at $q_0$. But, by applying an input symbol, say $a$, $M_N$ can reach any of the states in $\delta(q_0, a)$. To describe $M_N$ just after applying the input symbol $a$, we require all the possible states that $M_N$ can reach after the application of $a$. So, $M_D$ has to remember all these possible states at any instant of time. Hence, the states of $M_D$ are defined as subsets of $Q$. As $M_N$ starts with initial state $q_0$, $q_0'$ is defined as $[q_0]$. A string $u$ belongs to $L(M_N)$, if a final state is one of the possible states $M_N$ reaches on processing $u$. So, a final state in $M_D$ (i.e., an element of $F'$) is any subset of $Q$ containing some final state of $M_N$.

Now, we can define $\delta'$ as,

(iv) $\delta'([q_1, q_2, q_3, \ldots\ldots, q_i], a)$

$= \delta(q_1, a) \cup \delta(q_2, a) \cup \delta(q_3, a) \cup \ldots\ldots \cup \delta(q_i, a)$

Equivalently,

$$\delta'([q_1, q_2, q_3, \ldots\ldots, q_i], a) = [p_1 \ldots\ldots p_j]$$

if and only if

$$\delta(\{q_1, \ldots\ldots, q_i\}, a) = \{p_1, p_2, p_3 \ldots\ldots, p_j\}$$

Before proving $L = L(M_D)$ we prove an auxiliary result,

$$\delta'(q_0', x) = [q_1, \ldots\ldots\ldots\ldots, q_i]$$

if and only if $\delta(q_0, x) = \{q_1, \ldots\ldots, q_i\}$ for all $x$ in $\Sigma^*$ ...(i)

We prove by induction on $|x|$, the "if" part, i.e.,

$$\delta'(q'_0, x) = [q_1, q_2, \ldots\ldots, q_i], \text{ if } \delta(q_0, x) = \{q_1, q_2 \ldots\ldots q_i\} \quad \ldots(ii)$$

When $|x| = 0$, $\delta(q_0, \Lambda) = \{q_0\}$, and by definition of $\delta'$, $\delta'(q_0', \Lambda) = q_0' = [q_0]$. So, equation (ii) is true for $x$ with $|x| = 0$. Thus, there is a basis for induction. Assume that the equation (ii) is only true for all strings $y$ with $|y| \leq m$. Let $x$ be a string of length $m + 1$. We may write $x$ as $ya$, where $|y| = m$ and $a \in \Sigma$. Let $\delta(q_0, y) = \{p_1, \ldots\ldots, p_j\}$ and $\delta(q_0, ya) = \{r_1, r_2, r_3, \ldots\ldots r_k\}$. As $|y| \leq m$, then by inductive assumption, we have –

$$\delta'(q_0', y) = [p_1, \ldots\ldots\ldots, p_j] \quad \ldots\ldots(iii)$$

Also, $\{r_1, r_2, r_3, \ldots\ldots\ldots, r_k\}$

$= \delta(q_0, ya)$

$= \delta(\delta(q_0 y), a) = \delta(\{p_1, \ldots\ldots\ldots, p_j\}, a)$



By definition of $\delta'$ –

$$\delta'([p_1, \ldots\ldots\ldots p_j], a) = [r_1, r_2, \ldots\ldots\ldots, r_k] \quad \ldots\ldots(iv)$$

Hence,

$$\delta'(q_0', ya) = \delta'(\delta'(q'_0, y), a)$$

$= \delta'([p_1, \ldots\ldots, p_j], a) = [r_1, r_2, \ldots\ldots, r_k]$

[by equation (iii) and equation (iv)]

Thus we have proved –

$$\delta'(q_0', x) = (q_1, q_2, q_3, \ldots\ldots\ldots q_i],$$

for $x = ya$

By induction, this is true for all strings.

The other part (i.e., "only if") can be proved similarly.

Now, $x \in L(M_N)$ if and only if $\delta(q, x)$ contains a state of $F$. As we have shown in equation (i), $\delta(q_0, x)$ contains a state of $F$ if and only if $\delta'(q_0', x)$ is in $F'$. Hence $x \in L(M_N)$ if and only if $x \in L(M_D)$. The proves that DFA, $M_D$ accepts $L$.

***Q.13. Explain the finite automata with outputs. What are the differences between Moore machine and Mealy machine ?***

***Or***

***Formally define the following (with example) –***

***(i) Mealy machine (ii) Moore machine.***

***(R.G.P.V., June 2010)***

***Or***

***Write short note on automata with output.*** ***(R.G.P.V., Dec. 2004)***

***Or***

***Differentiate between Mealy and Moore machine. (R.G.P.V., Nov. 2018)***

***Ans.*** The finite automata have binary output, i.e., they accept the string or do not accept the string. This acceptability is decided on the basis of reachability of the final state from the initial state. This restriction can be removed by considering the model where the output can be chosen from some other alphabets. There are two distinct approaches, first in which the output is associated with the current state, called a ***Moore machine*** and second in which the output is associated with the transition, called a ***Mealy machine***. In other words, the value of the output function $Z(t)$ in the general case, is a function of the current state $q(t)$ and the present input $a(t)$, i.e.,

$$Z(t) = \lambda(q(t), a(t))$$

where $\lambda$ is called the output function. This generalized model is called ***Mealy machine***. If the output function $Z(t)$ depends only on the present state and is independent of the current input, the output function may be written as –

$$Z(t) = \lambda(q(t))$$

This restricted model is called ***Moore machine***. It is more convenient to use Moore machine in automata theory.

Formally, we define each as follows –

***(i) Moore Machine*** – The Moore machine is a six-tuple machine $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$, where

(a) $Q$ is a finite set of states

(b) $\Sigma$ is the input alphabet

(c) $\Delta$ is the output alphabet

(d) $\delta$ is the transition function which maps $\Sigma \times Q$ into $Q$

(e) $\lambda$ is the output function mapping $Q$ into $\Delta$

(f) $q_0$ is the initial state.

The $\lambda$ is a mapping from $Q$ to $\Delta$ giving the output associated with each state. The output of $M$ in response to input $a_1\ a_2 \ldots\ldots a_n$, $n \geq 0$ is $\lambda(q_0)\lambda(q_1)\ldots\ldots\lambda(q_n)$, where $q_0, q_1, \ldots\ldots q_n$ is the sequence of state such that $\delta(q_{i-1}, a_i) = q_i$ for $1 \leq i \leq n$. It is noted that machine gives output $\lambda(q_0)$ in response to input $\Lambda$. The DFA may be viewed as a special case of Moore machine where the output alphabet is {0,1} and state $q$ is "accepting" if and only if $\lambda(q) = 1$.

**Table 1.1 A Moore Machine**

| *Present State* | *Next State δ* | | *Output* |
|---|---|---|---|
| | *a = 0* | *a = 1* | *λ* |
| $\rightarrow q_0$ | $q_3$ | $q_1$ | 0 |
| $q_1$ | $q_1$ | $q_2$ | 1 |
| $q_2$ | $q_2$ | $q_3$ | 0 |
| $q_3$ | $q_3$ | $q_0$ | 0 |

For example, table 1.1 shows a Moore machine. The initial state $q_0$ is marked with an arrow. The table defines $\delta$ and $\lambda$. For the input string 0111, the transition of states is given by

$$q_0 \rightarrow q_3 \rightarrow q_0 \rightarrow q_1 \rightarrow q_2$$

The output string is 00010. For the input string $\Lambda$, the output is $\lambda(q_0) = 0$.

***(ii) Mealy Machine*** – The Mealy machine is also a six-tuple machine $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$, where all the symbols except $\lambda$ have the same meaning as in the Moore machine. $\lambda$ is the output function mapping $\Sigma \times Q$ into $\Delta$, i.e., $\lambda(q, a)$ gives the output associated with the transition from state $q$ on into $a$. The output of $M$ in response to input $a_1\ a_2 \ldots\ldots a_n$ is $\lambda(q_0, a_1)\lambda(q_1, a_2)\ldots\ldots\lambda(q_{n-1}, a_n)$, where $q_0, q_1, \ldots\ldots, q_n$ is the sequence of states such that $\delta(q_{i-1}, a_i) = q_i$ for $1 \leq i \leq n$. It is noted that this sequence has length $n$ rather than length $n + 1$ as for the Moore machine, and on input $\Lambda$ a Mealy machine gives output $\Lambda$.

For example, table 1.2 shows a Mealy machine which defines $\delta$ and $\lambda$. For the input string 0011, the transition of states is given by

$$q_1 \rightarrow q_3 \rightarrow q_2 \rightarrow q_4 \rightarrow q_3,$$



and the output string is 0100. In the case of Mealy machine, we get an output only on the application of an input symbol. So for input string $\Lambda$, the output is only $\Lambda$, while in the Moore machine the output is $\lambda(q_0)$.

**Table 1.2 A Mealy Machine**

| *Present State* | *Next State* | | | |
|---|---|---|---|---|
| | *a = 0* | | *a = 1* | |
| | *State* | *Output* | *State* | *Output* |
| $\rightarrow q_1$ | $q_3$ | 0 | $q_2$ | 0 |
| $q_2$ | $q_1$ | 1 | $q_4$ | 0 |
| $q_3$ | $q_2$ | 1 | $q_1$ | 1 |
| $q_4$ | $q_4$ | 1 | $q_3$ | 0 |

***Q.14. What do you mean by automata with output capability ? Draw a Mealy machine equivalent to the following circuit.***

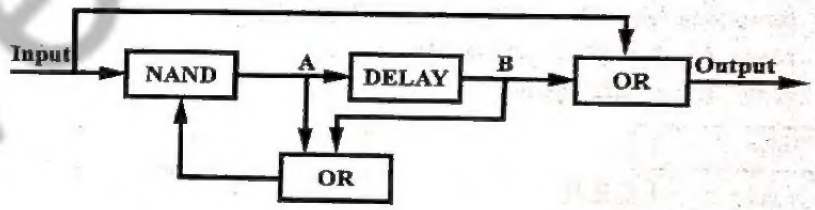



*Fig. 1.9*

***(R.G.P.V., Dec. 2015)***

***Ans.*** Refer to Q.13.

We denote current in wire by 1 and no current by 0. We identify four states based on whether current is available or not.

| *Q* | *Input* | |
|---|---|---|
| | *A* | *B* |
| $q_0$ | 0 | 0 |
| $q_1$ | 0 | 1 |
| $q_2$ | 1 | 0 |
| $q_3$ | 1 | 1 |

The operation of this circuit is such that after an input of 0's and 1's, the states changes according to the following rules –

New $B$ = Old A

New $A$ = (Input) NAND (old $A$ OR old $B$) (∵ D Flip flop)

Output = Input OR old $B$

Suppose we are in state $q_0$ and we are not receiving current i.e., input = 0

New $B$ = Old $A$ = 0 (Intable, $A = 0$, in state $q_0$)

New $B$ = 0

New $A$ = (Input) NAND (old $A$ OR old $B$)

New $A$ = 0 NAND (0 OR 0)

New $A$ = 0 NAND 0

New $A$ = 1

Output = Input OR old $B$

Output = 0 or 0

Output = 0

The next state is $q_2$ because, new $A = 1$ and new $B = 0$.

If we are in state $q_0$ and we receive current input = 1.

New $B$ = Old $A$ = 0 (from table, $A = 0$)

New $A$ = Input NAND (old $A$ OR old $B$)

New $A$ = 1 NAND (0 OR 0)

New $A$ = 1 NAND 0

New $A$ = 1

Output = 1 OR 0

Output = 1

The next state is $q_2$ (because new $A = 1$, new $B = 0$)

If we are in state $q_1$

Let, Input = 0 (No current)

New $B$ = Old $A$ = 0 (from table)

New $A$ = Input NAND (old $A$ OR old $B$)

New $A$ = 0 NAND 1

New $A$ = 1

Output = 0 OR 1

Output = 1

therefore next state is $q_2$ ($\because A = 1, B = 0$)

when Input = 1 (with current)

New $B$ = Old $A$ = 0

New $A$ = 1 NAND (0 OR 1)

New $A$ = 1 NAND 1



New $A$ = 0

Output = 1 OR 1

Output = 1

therefore next state is $q_0$ ($\because A = 0, B = 0$)

If we are in state $q_2$

when input = 0 (No current)

New $B$ = Old $A$ = 1 (from table)

New $A$ = 0 NAND (1 OR 0)

New $A$ = 0 NAND 1

New $A$ = 1

Output = 0 OR 0

Output = 0

therefore next state is $q_3$ ($\because A = 1, B = 1$)

when input = 1 (with current)

New $B$ = Old $A$ = 1

New $A$ = 1 NAND (1 OR 0)

New $A$ = 1 NAND 1

New $A$ = 0

Output = 1 OR 0

Output = 1

therefore next state is $q_1$ ($\because A = 0, B = 1$)

If we are in state $q_3$

when input = 0 (No current)

New $B$ = Old $A$ = 1

New $A$ = 0 NAND (1 OR 1)

New $A$ = 0 NAND 1

New $A$ = 1

Output = 0 OR 1

Output = 1

therefore next state is $q_3$ ($\because A = 1, B = 1$)

when input = 1

New $B$ = Old $A$ = 1

New $A = 1$ NAND (1 OR 1)

New $A = 1$ NAND 1

New $A = 0$

Output = 1 OR 1

Output = 1

therefore next state is $q_1$ ($\because A = 0, B = 1$)

**Table 1.3 Transition Table**

| Old State | Input = 0 | | Input = 1 | |
|---|---|---|---|---|
| | New State | Output | New State | Output |
| $q_0$ | $q_2$ | 0 | $q_2$ | 1 |
| $q_1$ | $q_2$ | 1 | $q_0$ | 1 |
| $q_2$ | $q_3$ | 0 | $q_1$ | 1 |
| $q_3$ | $q_3$ | 1 | $q_1$ | 1 |

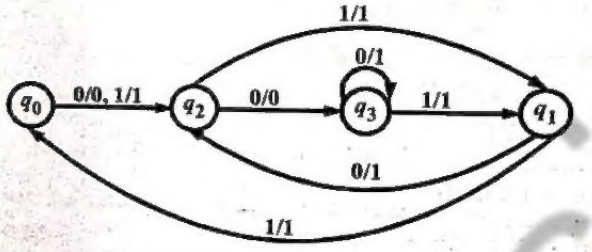



*Fig. 1.10 Transition Diagram*

***Q.15. Explain the procedure for transforming a Mealy machine into Moore machine with the help of an example.***

***Ans.*** Consider the Mealy machine described by the transition table given in table. 1.4.

**Table 1.4 An Example of Mealy Machine**

| Present State | Next State | | | |
|---|---|---|---|---|
| | Input $a = 0$ | | Input $a = 1$ | |
| | State | Output | State | Output |
| $\rightarrow q_1$ | $q_3$ | 0 | $q_2$ | 0 |
| $q_2$ | $q_1$ | 1 | $q_4$ | 0 |
| $q_3$ | $q_2$ | 1 | $q_1$ | 1 |
| $q_4$ | $q_4$ | 1 | $q_3$ | 0 |



The construction of the Moore machine which is equivalent to the Mealy machine is as follows –

At the first stage, we develop the procedure so that both machines accept exactly the same set of input sequences. We look into the next state column for any state, say $q_i$, and determine the number of different outputs associated with $q_i$ in that column.

Then we split $q_i$ into several different states, the number of such states being equal to the number of different outputs associated with $q_i$.

In the example in table 1.4, $q_1$ is associated with one output 1 and $q_2$ is associated with two different outputs 0 and 1. Similarly $q_3$ and $q_4$ are associated with the outputs 0 and 0, 1 respectively. So, we split $q_2$ and $q_4$ into $q_{20}$ and $q_{21}$, and $q_{40}$ and $q_{41}$ respectively. Table 1.5 can be reconstructed for the new states as in table 1.5.

**Table 1.5 Intermediate State Table**

| Present State | Next State | | | |
|---|---|---|---|---|
| | Input $a = 0$ | | Input $a = 1$ | |
| | State | Output | State | Output |
| $\rightarrow q_1$ | $q_3$ | 0 | $q_{20}$ | 0 |
| $q_{20}$ | $q_1$ | 1 | $q_{40}$ | 0 |
| $q_{21}$ | $q_1$ | 1 | $q_{40}$ | 0 |
| $q_3$ | $q_{21}$ | 1 | $q_1$ | 1 |
| $q_{40}$ | $q_{41}$ | 1 | $q_3$ | 0 |
| $q_{41}$ | $q_{41}$ | 1 | $q_3$ | 0 |

The pair of states and outputs in the next state column can be rearranged as given in table 1.6.

**Table 1.6 Rearranged State Table**

| Present State | Next State | | Output |
|---|---|---|---|
| | $a = 0$ | $a = 1$ | |
| $\rightarrow q_1$ | $q_3$ | $q_{20}$ | 1 |
| $q_{20}$ | $q_1$ | $q_{40}$ | 0 |
| $q_{21}$ | $q_1$ | $q_{40}$ | 1 |
| $q_3$ | $q_{21}$ | $q_1$ | 0 |
| $q_{40}$ | $q_{41}$ | $q_3$ | 0 |
| $q_{41}$ | $q_{41}$ | $q_3$ | 1 |

**Table 1.7 An Equivalent Moore Machine**

| Present State | Next State | | Output |
|---|---|---|---|
| | $a = 0$ | $a = 1$ | |
| $\rightarrow q_0$ | $q_3$ | $q_{20}$ | 0 |
| $q_1$ | $q_3$ | $q_{20}$ | 1 |
| $q_{20}$ | $q_1$ | $q_{40}$ | 0 |
| $q_{21}$ | $q_1$ | $q_{40}$ | 1 |
| $q_3$ | $q_{21}$ | $q_1$ | 0 |
| $q_{40}$ | $q_{41}$ | $q_3$ | 0 |
| $q_{41}$ | $q_{41}$ | $q_3$ | 1 |

Table 1.6 gives the Moore machine. Here, we observe that the initial state $q_1$ is associated with output 1. This means that with ...

of 1, if the machine starts at state $q_1$. Thus this Moore machine accepts zero-length sequence (null sequence) which is not accepted by the Mealy machine. To overcome this situation, either we must neglect the response of Moore machine to input $\Lambda$, or we must add a new starting state $q_0$, whose state transitions are identical with those of $q_1$ but whose output is 0. Thus table 1.6 is transformed as table 1.7.

It is clear from the procedure, if we have $m$-output, $n$-state Mealy machine, the corresponding $m$-output Moore machine has no more than $mn$ states.

***Q.16. Explain the procedure for transforming a Moore machine into a Mealy machine with the help of an example.***

***Ans.*** The following procedure is adopted to construct an equivalent Mealy machine from a given Moore machine.

(i) We have to define the output function $\lambda'$ for Mealy machine as a function of present state and input symbol. $\lambda'$ is defined as –

$\lambda'(q,a) = \lambda(\delta(q,a))$ for all states $q$ and input symbol $a$.

(ii) The transition function is the same as that of the given Moore machine.

**Table 1.8 An Example of Moore Machine**

| Present State | Next State | | Output |
|---|---|---|---|
| | $a=0$ | $a=1$ | |
| $\rightarrow q_0$ | $q_3$ | $q_1$ | 0 |
| $q_1$ | $q_1$ | $q_2$ | 1 |
| $q_2$ | $q_2$ | $q_3$ | 0 |
| $q_3$ | $q_3$ | $q_0$ | 0 |

**Table 1.9 An Equivalent Mealy Machine**

| Present State | Next State | | | |
|---|---|---|---|---|
| | Input $a=0$ | | Input $a=1$ | |
| | State | Output | State | Output |
| $\rightarrow q_0$ | $q_3$ | 0 | $q_1$ | 1 |
| $q_1$ | $q_1$ | 1 | $q_2$ | 0 |
| $q_2$ | $q_2$ | 0 | $q_3$ | 0 |
| $q_3$ | $q_3$ | 0 | $q_0$ | 0 |

Consider the Moore machine described by the transition table given in table 1.8.

We must follow the reverse procedure of converting Mealy machine to Moore machine. In case of the Moore machine, for every input symbol we form the pair consisting of the next state and the corresponding output and reconstruct the table for Mealy machine.

For example, in table 1.8 the states $q_3$ and $q_1$ in the next state column should be associated with outputs 0 and 1, respectively. The transition table for Mealy machine is given in table 1.9.

It is noted that the number of states can be reduced in any model by considering states with identical transition. If two states have identical transition



(i.e., the rows corresponding to the two-states are identical), then we can remove one of them.

***Q.17. What is Moore machine ? How finite automata can be converted into Moore machine ? Explain with the help of example.***
***(R.G.P.V., June 2015)***

***Ans.*** Refer to Q.13.

A finite automata can be converted into a Moore machine by introducing output alphabet $\Delta = \{0, 1\}$ and defining the output function $\lambda(q)$, such that

$$\lambda(q) = \begin{cases} 1 & \text{if } q \in F \\ 0 & \text{if } q \notin F \end{cases}$$

For example, consider the automata given by table 1.10.

**Table 1.10 Finite Automata**

| Present State | Inputs | |
|---|---|---|
| | $a=0$ | $x=1$ |
| $\rightarrow q_0$ | $q_2$ | $q_1$ |
| $q_1$ | $q_3$ | $q_0$ |
| $q_2$ | $q_0$ | $q_3$ |
| $q_3$ | $q_1$ | $q_2$ |

**Table 1.11 Moore Machine**

| Present State | Next State | | Output |
|---|---|---|---|
| | $a=0$ | $x=1$ | |
| $\rightarrow q_0$ | $q_2$ | $q_1$ | 1 |
| $q_1$ | $q_3$ | $q_0$ | 0 |
| $q_2$ | $q_0$ | $q_3$ | 0 |
| $q_3$ | $q_1$ | $q_2$ | 0 |

Let $\Delta = \{0, 1\}$ be the introduced alphabet. The value of output is defined by output function $\lambda(q)$, we have

$$\lambda(q_0) = 1,\ \lambda(q_1) = 0,\ \lambda(q_2) = 0,\ \lambda(q_3) = 0,$$

The Moore machine is constructed by writing output values calculated corresponding to states of FA. The Moore machine is shown in table 1.11.

## NUMERICAL PROBLEMS

***Prob.1. Design a FA which accepts set of strings containing four 1's in every string over alphabet $\Sigma = \{0, 1\}$.*** ***(R.G.P.V., Dec. 2008)***

***Sol.*** The finite automaton $M$ is given by

$$M = (\{q_0, q_1, q_2, q_3, q_4, q_5\}, \{0, 1\}, \delta, q_0, \{q_4\})$$

The finite automaton is as shown in fig. 1.11.

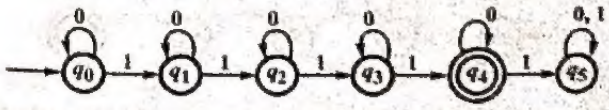



***Fig. 1.11 Finite Automaton***

***Prob.2. Design DFA that accepts all strings with at most 3 a's.*** ***(R.G.P.V., June 20***

***Sol.*** The finite automaton $M$ is given by

$$M = (\{q_0, q_1, q_2, q_3, q_4\}, (a, b), \delta, q_0, \{q_0, q_1, q_2, q_3\})$$

The finite automaton is shown in fig. 1.12.

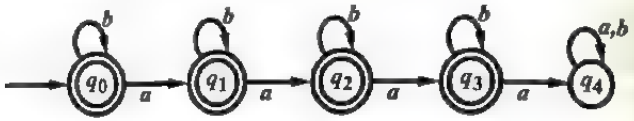



***Fig. 1.12 DFA***

***Prob.3. Construct a finite automata for the language $\{0^n \mid n \bmod 3 = 2, n \geq 0\}$.*** ***(R.G.P.V., Dec. 20***

***Sol.*** No. of strings generated by language are –

00, 00000, 00000000,00000000000, .....

So, finite automata for the given language is shown in fig. 1.13.

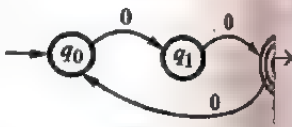



***Fig. 1.13***

***Prob.4. Design a NFA for $\{cbab^n \mid n >= 0\}$.*** ***(R.G.P.V., June 20***

***Sol.*** NFA for the language

$$L = \{cbab^n \mid n >= 0\}$$

is $M = (\{q_0, q_1, q_2, q_3\}, \{a, b, c\}, \delta, q_0, \{q_3\})$

Here, the NFA is such that it accepts all strings of the type *cba, cbabb, cbabbb,* .....

So, NFA for the given language is shown in fig. 1.14.

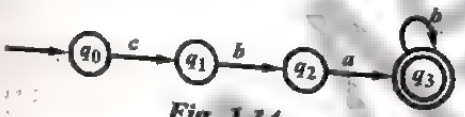



***Fig. 1.14***

***Prob.5. Draw a deterministic finite automata for alphabet {a, b}, the language {w/w contains the substring abab}.*** ***(R.G.P.V., Dec. 20***

***Sol.*** Given language is,

$L$ = {$w$/$w$ contains the substring *abab*}

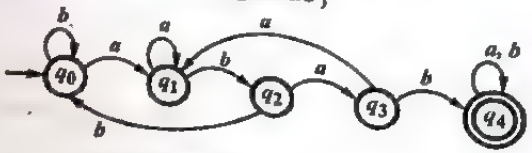



***Fig. 1.15 Finite Automata Accepting {a, b}* {abab}***



***Prob.6. Design a machine which checks whether a given decimal number is "EVEN".***

***Sol.*** There will be two states one for EVEN and second one for odd. Symbols, 0, 2, 4, 6, 8 indicate that they are even numbers because when we divide these numbers by '2' we get remainder '0' otherwise odd number.

**Table 1.12 State Table**

| State \ Input | 0, 2, 4, 6, 8 | 1, 3, 5, 7, 9 |
|---|---|---|
| Starting state → $q_0$ | $q_0$ | $q_1$ |
| $q_1$ odd | $q_2$ | $q_1$ |
| ($q_2$) even | $q_2$ | $q_1$ |

**Note** – '→' This symbol shows that given state is starting state i.e., → $q_0$ and Ⓞ $q_0$

'◯' Final state which is accepting state.

Therefore, $\Sigma$ = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

$Q = \{q_0, q_1, q_2\}$

$\delta$ = The transition of states

$q_0$ = Start state

$F = \{q_2\}$

M.A.F. →

**Table 1.13**

| Q \ Σ | 0, 2, 4, 6, 8 | 1, 3, 5, 7, 9 |
|---|---|---|
| $q_0$ | 1 | 0 |
| $q_1$ | 1 | 0 |
| $q_2$ | 0 | 0 |

1 → even

0 → 0

**Transition Diagram** –

Suppose the number is – 246710321

$\underline{q_0}$ 246710321

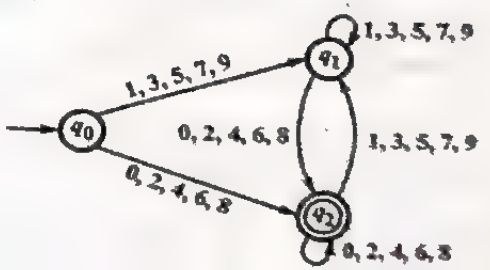
2 $\underline{q_2}$ 46710321

24 $\underline{q_2}$ 6710321

246 $\underline{q_2}$ 710321

2467 $\underline{q_1}$ 10321

24671 $\underline{q_1}$ 0321



246710 $q_2$ 321

2467103 $q_1$ 21

24671032 $q_2$ 1

246710 321 $q_1$

As we end up in the state $q_1$ which is a non-final state therefore g number is odd.

***Prob.7. Design a machine which can check that given string is divisi by '4'.***

***Sol.*** To check divisibility by '4' there will be four states with remain '0', remainder '1', remainder '2' and remainder '3' and one more state starting state.

**Table 1.14**

| Q \ Σ | 0, 4, 8 | 1, 5, 9 | 2, 6 | 3, 7 |
|---|---|---|---|---|
| S | $q_0$ | $q_1$ | $q_2$ | $q_3$ |
| Rem '0' $q_0$ | $q_0$ | $q_1$ | $q_2$ | $q_3$ |
| Rem '1' $q_1$ | $q_2$ | $q_3$ | $q_0$ | $q_1$ |
| Rem '2' $q_2$ | $q_0$ | $q_1$ | $q_2$ | $q_3$ |
| Rem '3' $q_3$ | $q_2$ | $q_3$ | $q_0$ | $q_1$ |

**Note** – We can merge two states by one, if both states have same transiti and both of them are final or non-final states.

In table 1.14 row-$q_1$ and row-$q_3$ both are identical and both of them non-final states. So we are replacing state $q_3$ by $q_1$

**Table 1.15**

| Q \ Σ | 0, 4, 8 | 1, 5, 9 | 2, 6 | 3, 7 |
|---|---|---|---|---|
| Start | $q_0$ | $q_{13}$ | $q_2$ | $q_{13}$ |
| $q_0$ | $q_0$ | $q_{13}$ | $q_2$ | $q_{13}$ |
| $q_{13}$ | $q_2$ | $q_{13}$ | $q_0$ | $q_{13}$ |
| $q_2$ | $q_0$ | $q_{13}$ | $q_2$ | $q_{13}$ |

we can observe that two columns in table 1.15 are same i.e., for 1, 5, 9 and 7. Hence they can be grouped into one.

**Table 1.16**

| Q \ Σ | 0, 4, 8 | 1, 3, 5, 7, 9 | 2, 6 |
|---|---|---|---|
| Start (S) | $q_0$ | $q_{13}$ | $q_2$ |
| $q_0$ | $q_0$ | $q_{13}$ | $q_2$ |
| $q_{13}$ | $q_2$ | $q_{13}$ | $q_0$ |
| $q_2$ | $q_0$ | $q_{13}$ | $q_2$ |



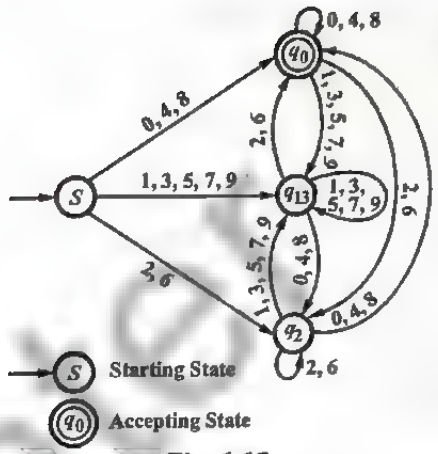



*Fig. 1.17*

***Prob.8. Construct an deterministic accepter equivalent to $M_N$ = $\{q_0, q_1\}$, {a, b}, δ, $q_0$, $\{q_0\}$). δ is defined by its state table given in table 1.17.***

**Table 1.17 State Table of $M_N$**

| State | Input | |
|---|---|---|
| | a | b |
| →$q_0$ | $q_0$ | $q_1$ |
| $q_1$ | $q_1$ | $q_0$,$q_1$ |

***Sol.*** For the deterministic accepter $M_D$ –

(i) The states are subsets of $\{q_0, q_1\}$, i.e., ϕ, $[q_0]$, $[q_0, q_1]$, $[q_1]$.

(ii) $[q_0]$ is the initial state.

(iii) $[q_0]$ and $[q_0, q_1]$ are the final states, as they are the only states having $[q_0]$.

(iv) δ is defined by the state table given in table 1.18.

**Table 1.18 State Table of $M_D$**

| State | Input | |
|---|---|---|
| | a | b |
| ϕ | ϕ | ϕ |
| $[q_0]$ | $[q_0]$ | $[q_1]$ |
| $[q_1]$ | $[q_1]$ | $[q_0,q_1]$ |
| $[q_0,q_1]$ | $[q_0,q_1]$ | $[q_0,q_1]$ |

$q_0$ and $q_1$ appear in the rows corresponding to $q_0$ and $q_1$ and the column corresponding to $a$. So, $\delta([q_0, q_1], a) = [q_0, q_1]$.

When $M_N$ has $n$ states, the corresponding finite accepter has $2^n$ st
However, we need not construct $\delta$ for all these states, but only for th
states reachable from $[q_0]$, this is because, we have to construct only
accepting $L(M_N)$. So, we start the construction of $\delta$ for $[q_0]$. We proceed
considering only states appearing earlier under input columns and constru
$\delta$ for them. We halt when no more new states appear under the input colu

***Prob.9. Give DFA accepting the language over alphabet {0, 1} a
that all strings of 0 and 1 ending in 101.*** ***(R.G.P.V., Dec. 2***

***Sol.*** The expression (0 + 1)*101 denotes all strings of 0's and 1's en
in 101. Now we construct a DFA accepting the regular expression (0, 1)*
corresponding to given language. The set of input alphabet is {0, 1}.

Fig. 1.18 shows a transition diagram for the FA.

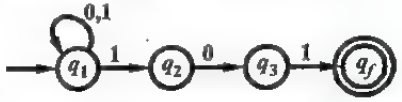



***Fig. 1.18 FA Accepting (0 + 1)*101***

Now, we construct a transition table for fig. 1.18.

**Table 1.19 Transition Table for FA**

| *State* | *0* | *1* |
|---|---|---|
| $q_1$ | $q_1$ | $q_1 q_2$ |
| $q_2$ | $q_3$ | – |
| $q_3$ | – | $q_f$ |
| $q_f$ | – | – |

Now, we convert above NFA into DFA as –

**Table 1.20 Transition Table for DFA**

| *State* | *0* | *1* |
|---|---|---|
| $[q_1]$ | $[q_1]$ | $[q_1\, q_2]$ |
| $[q_1\, q_2]$ | $[q_1\, q_3]$ | $[q_1\, q_2]$ |
| $[q_1\, q_3]$ | $[q_1]$ | $[q_1\, q_2\, q_f]$ |
| $[q_1\, q_2\, q_f]$ | $[q_1\, q_3]$ | $[q_1\, q_2]$ |

State diagram for required DFA which accepted strings ending in 10
shown in fig. 1.19.

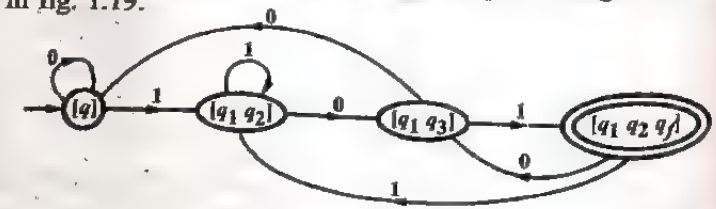



***Fig. 1.19 DFA Accepting Strings Ending in 101***



***Prob.10. Design deterininistic finite automaton accepting the following
languages over the alphabet {0, 1} –***

***(i) The set of all words ending in 00.***

***(ii) The set of all words except ∈.***

***(iii) The set of all words that begin with 0.***

***(R.G.P.V., Dec. 2015)***

Sol. (i) The expression (0 + 1)* 00 denotes all strings of 0's and 1's
nding in 00. Fig. 1.20 shows a transition diagram for the DFA.

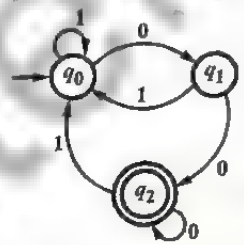



***Fig. 1.20***

(ii) The DFA accepting the set of all words except ∈ over the alphabet
0, 1} is shown in fig. 1.21.

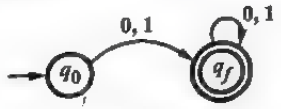



***Fig. 1.21***

(iii) The DFA accepting the set of all words that begin with 0 over
he alphabet {0,1} is shown in fig. 1.22.

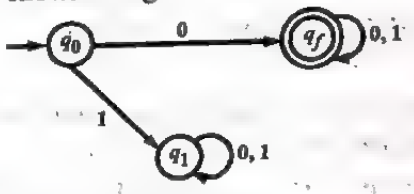



***Fig. 1.22***

***Prob.11. Design DFA accepting the language over the alphabet 0, 1
hat have the set of all strings ending in 00.*** ***(R.G.P.V., Dec. 2016)***

***Sol.*** Refer to the sol. of Prob.10 (i).

***Prob.12. Design FA which accepts even no. of 0's and even no. of 1's.***
***(R.G.P.V., Dec. 2011)***

***Or***

***Construct DFA over input alphabet Σ = {0, 1} to accept string which
ontains no. of 0 is even and no. of 1 is even.*** ***(R.G.P.V., June 2010)***

***Or***

***Construct a DFA accepting set of all strings containing even no. of a and even no. of b's over alphabet {a, b}.*** ***(R.G.P.V., June 2***

***Sol.*** Consider the transition diagram of fig. 1.23. This FA is denoted $M = (Q, \Sigma, \delta, q_0, F)$, where $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{a, b\}$, $F = \{q_0\}$, $\delta$ is shown in fig. 1.24.

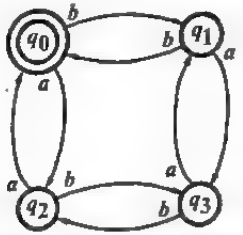



***Fig. 1.23 The Transition Diagram of a Finite Automaton***

| States | Inputs a | Inputs b |
|---|---|---|
| $q_0$ | $q_2$ | $q_1$ |
| $q_1$ | $q_3$ | $q_0$ |
| $q_2$ | $q_0$ | $q_3$ |
| $q_3$ | $q_1$ | $q_2$ |

***Fig. 1.24 $\delta$ for the FA of Fig.1***

Suppose *bbabab* is input to *M*. Since $\delta(q_0, b) = q_1$ and $\delta(q_1, b) = q_0$, $\delta(q_0, bb) = \delta(\delta(q_0, b), b) = \delta(q_1, b) = q_0$

Therefore, *bb* is in $L(M)$, but we are interested in *bbabab*. We continu $\delta(q_0, a) = q_2$. Thus,

$$\delta(q_0, bba) = \delta(\delta(q_0, bb), a) = \delta(q_0, a) = q_2$$

Continuing in this manner, we find that

$$\delta(q_0, bbab) = q_3, \ \delta(q_0, bbaba) = q_1$$

and finally $\delta(q_0, bbabab) = q_0$

The entire state of sequence is

$$q_0 \overset{b}{} q_1 \overset{b}{} q_0 \overset{a}{} q_2 \overset{b}{} q_3 \overset{a}{} q_1 \overset{b}{} q_0$$

Thus *bbabab* is in $L(M)$.

***Prob.13. Construct a finite automaton accepting all strings over {0,***

***(i) Having odd number of 0's***

***(ii) Having even number of 0's and even number of 1's.***

***Sol. (i) Having odd number of 0's –***

The finite automaton *M* is given by

$$M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$$


***Fig. 1.25***

***(ii) Having even number of 0's and even number of 1's –*** Refe Prob.12.



***Prob.14. Construct DFA for –***

***(i) Binary integer divisible by 3 (ii) $a^n b \mid n \geq 0$.***

***(R.G.P.V., Dec. 2009)***

***Sol.*** (i) In binary number when we vant to find the corresponding value eceiving a new bit then remember the riginal value is doubled and new bit is dded.

In starting state if we receive '0', hen we go to state where remainder is 0', if we get '1' then the next state is emainder '1'.

**Table 1.21**

| Q \ Σ | 0 | 1 |
|---|---|---|
| Start → (S) | $q_0$ | $q_1$ |
| (rem '0') ($q_0$) | $q_0$ | $q_1$ |
| (rem '1') $q_1$ | $q_2$ | $q_0$ |
| (rem '2') $q_2$ | $q_1$ | $q_2$ |

Formula – 2 × Old remainder + Next bit

In state $q_0$ – 2 × 0 (rem '0') + 0 (next bit)
= 0 + 0
= 0 (remainder '0', i.e. state $q_0$)
2 × 0 (rem '0') + 1 (next bit)
= 0 + 1 = 1

In state $q_1$ – 2 × 1 (rem '1') + 0 (next bit)
= 2 + 0 = 2
2 × 1 (rem '1') + 1(next bit)
= 2 + 1 = 3

is divisible by '3' and remainder is '0' i.e., state '$q_0$'

In state $q_2$ – 2 × 2 (old remainder) + 0 (next bit)
= 4 + 0
= 4 (remainder '1' i.e., $q_1$)
2 × 2 (old remainder '2') + 1(next bit)
= 4 + 1
= 5 (remainder '2' i.e., state $q_2$)

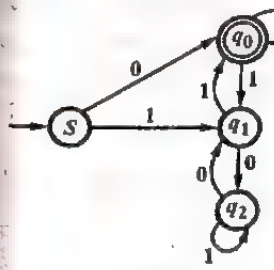



***Fig. 1.26***


***Fig. 1.27***

(ii) No. of strings generated by language $a^n b | n \geq 0$ are– *b, ab, aab, aab, aaaab,* .....

DFA for the given language is shown in fig. 1.27.

***Prob.15. Construct a finite automaton that will accept those strings a binary number that are divisible by three.*** ***(R.G.P.V., Dec. 2***

***Sol.*** Refer to Prob.14 (i).

***Prob.16. Define deterministic finite automata. Draw DFA that acc any string which ends with 1 or it ends with an even number of 0's follow the last 1. Alphabets are {0, 1}.*** ***(R.G.P.V., Dec. 2***

***Sol.*** **DFA** – Refer to Q.7.

A *DFA* can be determined as,

$$M = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_1\})$$

where $\delta$ is defined in form of transition table 1.22.

Class of language accepted by *DFA* is

$$L = \{0, 1\}^* \{1\} \text{ or } \{0, 1\}^*\{1\}\ \{0, 0\}^*$$

**Table 1.22**

| State | Inpu | |
|---|---|---|
| | 0 | 1 |
| → $q_0$ | $q_0$ | $q_1$ |
| ($q_1$) | $q_2$ | $q_1$ |
| $q_2$ | $q_1$ | $q_2$ |


***Fig. 1.28***

***Prob.17. Construct a DFA accepting all strings w over {0, 1} such the number of 1's in w is 3 mod 4.*** ***(R.G.P.V., Dec. 2***

***Sol.*** The finite automata M is given by

$$M = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_3\})$$

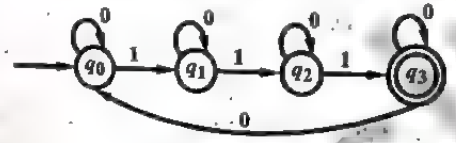



***Fig. 1.29***

***Prob.18. Construct NFA for the following grammar –***

***$S \rightarrow Ab|ab, A \rightarrow Ab|Bb, B \rightarrow Ba|a$*** ***(R.G.P.V., June 2***

***Sol.*** The above grammar is left linear grammar which accepts the lang of the form $\{a^n b^m \mid n, m \geq 1\}$. It should be converted to right linear gram When the above grammar is converted to right linear grammar. It is represent

$S \rightarrow bA|ba, A \rightarrow bA|bB, B \rightarrow aB|a$

Let $M = (Q, \Sigma, \delta[S], [\in])$, where $Q = \{[S], [bA], [A], [ba], [a], [$ $[B], [aB], [\in]\}$, $T = \{a, b\}$ and $\in$ is

$\delta([S], \in) = [bA], [ba]$

$\delta([A], \in) = [bA], [bB]$

$\delta([B], \in) = [aB], [a]$

$\delta([bA], b) = [A]$

$\delta([ba], b) = [a]$



$\delta([a], a) = [\in]$

$\delta([bB], b) = [B]$

$\delta([aB], a) = [B]$

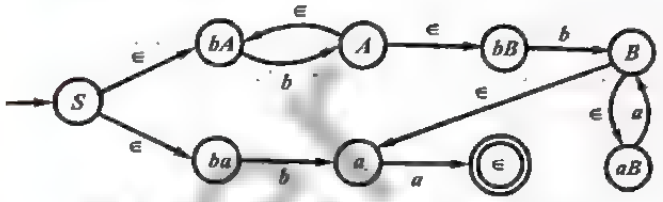



***Fig. 1.30 NFA for Right Linear Grammar***

The NFA for right linear grammar is shown in fig. 1.30. To get the final NFA for the given left linear grammar, we reverse the NFA and the result obtained is shown in fig. 1.31.

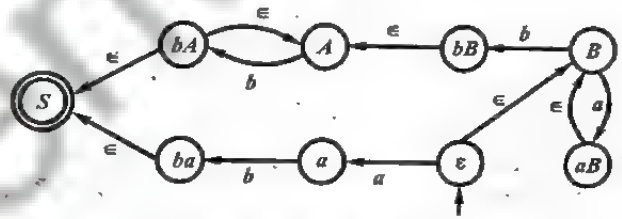



***Fig. 1.31 NFA for Left Linear Grammar***

***Prob.19. Construct an DFA accepting the set of all string over the alphabet {0, 1}, such that number of 0's divisible by 5 and number of 1's divisible by 3.*** ***(R.G.P.V., Dec. 2012)***

***Or***

***Design a DFA that accepts the string such that number of zero divisible by five and number of one divisible by three.*** ***(R.G.P.V., Dec. 2013)***

***Sol.*** Consider the transition diagram of fig. 1.32. This FA is denoted by $M = (Q, \Sigma, \delta, q_0, F)$, where $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}, q_{11}, q_{12}, q_{13}, q_{14}\}$ and $F = \{q_0\}$, $\Sigma = \{0, 1\}$ and $\delta$ is shown in table 1.23.

**Table 1.23 Transition Table**

| Q \ Σ | 0 | 1 |
|---|---|---|
| First Zero $q_0$ | $q_1$ | $q_5$ First One |
| Second Zero $q_1$ | $q_2$ | $q_7$ First One |
| Third Zero $q_2$ | $q_3$ | $q_9$ First One |
| Fourth Zero $q_3$ | $q_4$ | $q_{11}$ First One |
| Fifth Zero $q_4$ | ($q_0$) | $q_{13}$ First One |
| Second One $q_5$ | $q_7$ | $q_6$ |

| | | | |
|---|---|---|---|
| Third One | $q_6$ | $q_8$ | $q_0$ |
| Second One | $q_7$ | $q_9$ | $q_8$ |
| Third One | $q_8$ | $q_{10}$ | $q_1$ |
| Second One | $q_9$ | $q_{11}$ | $q_{10}$ |
| Third One | $q_{10}$ | $q_{12}$ | $q_2$ |
| Second One | $q_{11}$ | $q_{13}$ | $q_{12}$ |
| Third One | $q_{12}$ | $q_{14}$ | $q_3$ |
| Second One | $q_{13}$ | $q_5$ | $q_{14}$ |
| Third One | $q_{14}$ | $q_6$ | $q_4$ |

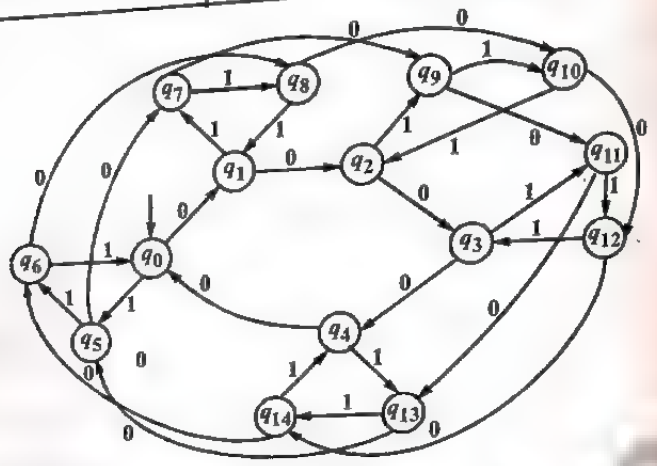



*Fig. 1.32*

***Prob.20. Construct DFA equivalent to the NFA.***

$M = (\{p, q, r, s\}, \{0, 1\}, \delta, p, \{q, s\})$

***where δ is defined in the following table –***

| δ | 0 | 1 |
|---|---|---|
| p | {q, s} | {q} |
| q | {r} | {q, r} |
| r | {s} | {p} |
| s | – | {p} |

***(R.G.P.V., Dec. 2***

Sol. The DFA $M_1$ equivalent to $M$ is defined as follows –

$$M_1 = \{2^Q, \{0, 1\}, \delta, p, F\}$$

where, $F = \{[q, s], [p, q, s], [r, q, s], [p, q, r, s]\}$

We start the construction by considering $[p]$ first. The state table is g in table 1.24.

**Table 1.24 State Table of DFA**

| State | 0 | 1 |
|---|---|---|
| $[p]$ | $[q,s]$ | $[q]$ |
| $[q]$ | $[r]$ | $[q,r]$ |
| $[q,s]$ | $[r]$ | $[q,r,p]$ |
| $[r]$ | $[s]$ | $[p]$ |
| $[q,r]$ | $[r,s]$ | $[q,r,p]$ |
| $[q,r,p]$ | $[q,s,r]$ | $[p,q,r]$ |
| $[s]$ | $\phi$ | $[p]$ |
| $[r,s]$ | $[s]$ | $[p]$ |
| $[q,s,r]$ | $[r,s]$ | $[p,q,r]$ |

***Prob.21. Construct a DFA equivalent to an NDFA whose transition table is defined by***

| State | a | b |
|---|---|---|
| $q_0$ | $q_1, q_3$ | $q_2, q_3$ |
| $q_1$ | $q_1$ | $q_3$ |
| $q_2$ | $q_3$ | $q_2$ |
| $q_3$ | – | – |

***(R.G.P.V., Dec. 2017)***

***Sol.*** The deterministic automaton $M_1$ equivalent to M is defined as follows –

$$M_1 = (2^Q, \{a, b\}, \delta, [q_0], F')$$

where,

We start the construction by considering $[q_0]$ first. We get $[q_1, q_3]$ and $[q_2, q_3]$. Then we construct δ for $[q_1, q_3]$ and $[q_2, q_3]$. After constructing δ for $[q_1, q_3]$ and $[q_2, q_3]$, we do not get any new states and so we terminate the construction of δ. The state table is given in table 1.25.

**Table 1.25 State Table of DFA**

| State | a | b |
|---|---|---|
| $[q_0]$ | $[q_1,q_3]$ | $[q_2,q_3]$ |
| $[q_1,q_3]$ | $[q_1]$ | $[q_3]$ |
| $[q_1]$ | $[q_1]$ | $[q_3]$ |
| $[q_2,q_3]$ | $[q_3]$ | $[q_2]$ |
| $[q_2]$ | $[q_3]$ | $[q_2]$ |
| $[q_3]$ | $\phi$ | $\phi$ |

***Prob.22. Convert the following NFA into an equivalent deterministic machine –***

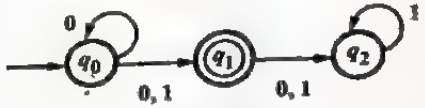



*Fig. 1.33*

***Sol.*** The *NFA* starts in state $q_0$, so the initial state of the *DFA* wil labelled $\{q_0\}$. Since $\delta_N(q_0, 0) = \{q_0, q_1\}$, we introduce the state $\{q_0, q_1\}$ in

$G_D$ and add an edge labeled 0 between $\{q_0\}$ and $\{q_0, q_1\}$. In the same way considering $\delta_N(q_0, 1) = \{q_1\}$ gives us the new state $\{q_1\}$ and an edge labeled 1 between it and $\{q_0\}$.

Similarly,

$$\delta_N(\{q_1\}, 0) = \{q_2\}$$

$$\delta_N(\{q_1\}, 1) = \{q_2\}$$

$$\delta_N(\{q_0, q_1\}, 0) = \delta(\{q_0\}, 0) \cup \delta(\{q_1\}, 0)$$
$$= \{q_0, q_1\} \cup \{q_2\} = \{q_0, q_1, q_2\}$$

$$\delta_N(\{q_0, q_1, q_2\}, 0) = \delta(\{q_0\}, 0) \cup \delta(\{q_1\}, 0) \cup \delta(\{q_2\}, 0)$$
$$= \{q_0, q_1,\} \cup \{q_2\} \cup \phi = \{q_0, q_1, q_2\}$$

$$\delta_N(\{q_0, q_1, q_2\}, 1) = \{q_1, q_2\}$$

$$\delta_N(\{q_0, q_1,\}, 1) = \delta(\{q_0\}, 1) \cup \delta(\{q_1\}, 1)$$
$$= \{q_1\} \cup \{q_2\} = \{q_1, q_2\}$$

$$\delta_N(\{q_1, q_2\}, 0) = \{q_2\}$$

$$\delta_N(\{q_1, q_2\}, 1) = \{q_2\}$$

$$\delta_N(\{q_2\}, 0) = \phi$$

$$\delta_N(\{q_2\}, 1) = \{q_2\}$$

The result, shown in fig. 1.34, is a DFA, equivalent to the NFA.

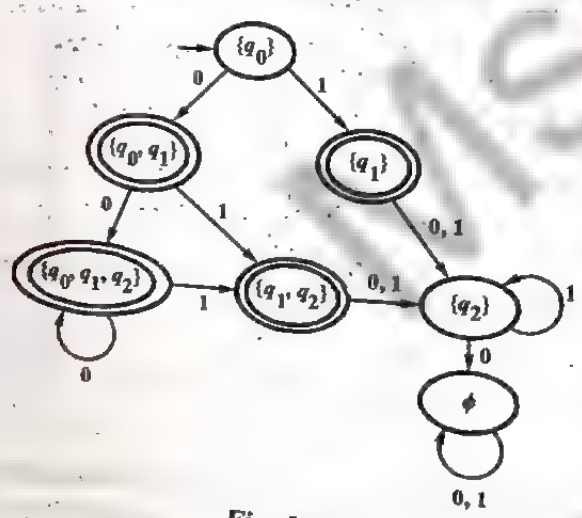



*Fig. 1.34*



***Prob.23. Construct minimized DFA for the given NFA.***

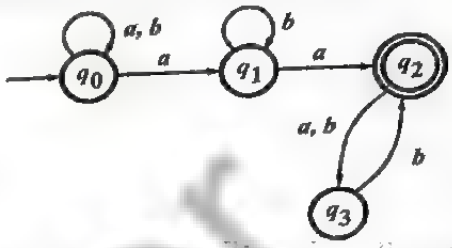



*Fig. 1.35* ***(R.G.P.V., June 2009)***

***Or***

***Convert the following NFA into DFA (see fig. 1.35). (R.G.P.V., Dec. 2008)***

***Sol.*** **Construction of DFA** – Table 1.26 shows the transition table of the given NFA.

**Table 1.26 Transition Table**

| State | Input | |
|---|---|---|
| | a | b |
| → $q_0$ | $q_0, q_1$ | $q_0$ |
| $q_1$ | $q_2$ | $q_1$ |
| ($q_2$) | $q_3$ | $q_3$ |
| $q_3$ | – | $q_2$ |

**Table 1.27 Transition Table for the DFA**

| State | Input | |
|---|---|---|
| | a | b |
| → $[q_0]$ | $[q_0, q_1]$ | $[q_0]$ |
| $[q_0, q_1]$ | $[q_0, q_1, q_2]$ | $[q_0, q_1]$ |
| $[q_0, q_1, q_2]$ | $[q_0, q_1, q_2, q_3]$ | $[q_0, q_1, q_3]$ |
| $[q_0, q_1, q_2, q_3]$ | $[q_0, q_1, q_2, q_3]$ | $[q_0, q_1, q_2, q_3]$ |
| $[q_0, q_1, q_3]$ | $[q_0, q_1, q_2]$ | $[q_0, q_1, q_2]$ |

The successor table is constructed for the DFA, and given in table 1.27. Fig. 1.36 shows the transition diagram reduced DFA.

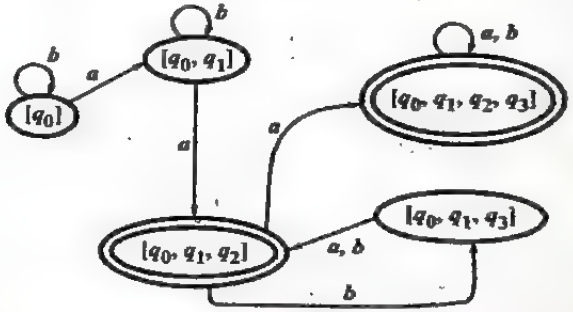



*Fig. 1.36*

***Prob.24. Convert the following NFA with ∈ to equivalent DFA.***

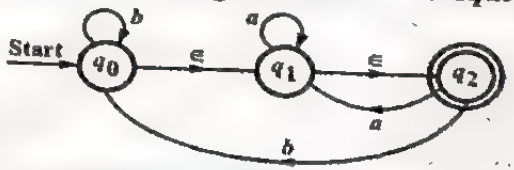



*Fig. 1.37* ***(R.G.P.V., June 2011)***

**Sol.** The conversion of NDFA to DFA is given step-by-step in fig. 1.

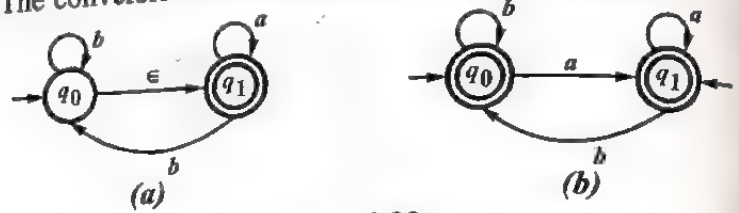



*Fig. 1.38*

The DFA shown in fig. 1.38 (b) can be redrawn as fig. 1.39.


*Fig. 1.39*

***Prob.25. Convert the following NDFA to deterministic finite automat***

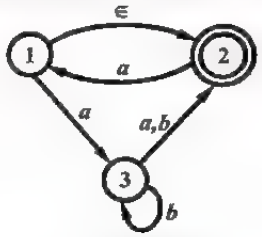



*Fig. 1.40* ***(R.G.P.V., Dec. 20***

**Sol.** We construct the transition table corresponding to NDFA with ∈-moves as given in table 1.28.

**Table 1.28**

| State | Input | |
|---|---|---|
| | a | b |
| ① | 1, 3 | – |
| ② | 1 | – |
| 3 | 2 | 3, 2 |

**Table 1.29 Transition Table for the DI**

| State | Input | |
|---|---|---|
| | a | b |
| [1] | [1, 3] | ϕ |
| [1, 3] | [1, 2, 3] | [2, 3] |
| [2, 3] | [1, 2] | [2, 3] |
| [1, 2] | [1, 3] | ϕ |
| [1, 2, 3] | [1, 2, 3] | [2, 3] |

The successor table is constructed for DFA and given in table 1.29.

The state diagram for the successor table is the required DFA as indicated in fig. 1.41 since ① and ② are the only final states of NDFA, therefore all states are the final states of DFA.

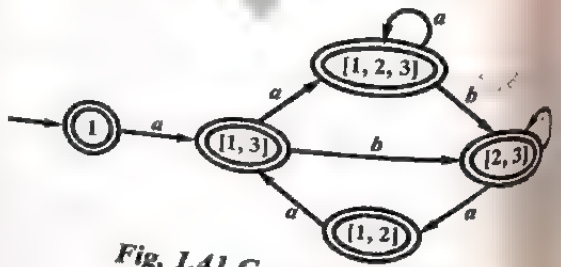



*Fig. 1.41 Constructed DFA*



***Prob.26. Determine the DFA equivalent to the given NFA –***

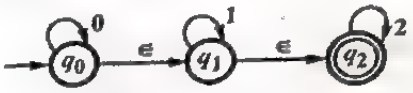



*Fig. 1.42* ***(R.G.P.V., Dec. 2016)***

Sol. We know that

∈-closure $(q)$ = set of all vertices $p$ such that there is a path from $q$ to $p$ labelled ∈.

and $\hat{\delta}(q, \in) = \in$-closure $(q)$

So, $\hat{\delta}(q_0, \in) = \in$-closure $(q_0) = \{q_0, q_1, q_2\}$

$\hat{\delta}(q_1, \in) = \in$-closure $(q_1) = \{q_1, q_2\}$

$\hat{\delta}(q_2, \in) = \in$-closure $(q_2) = \{q_2\}$

Let $\{q_0, q_1, q_2\} = A_1$, $\{q_1, q_2\} = A_2$ and $\{q_2\} = A_3$

So, $\hat{\delta}(A_1, 0) = \in$-closure $(\delta(\hat{\delta}(A_1, \in), 0)$

$= \in$-closure $(\delta(\hat{\delta}(q_0, \in) \cup \hat{\delta}(q_1, \in) \cup \hat{\delta}(q_2, \in), 0))$

$= \in$-closure $(\delta(A_1 \cup A_2 \cup A_3, 0))$

$= \in$-closure $(\delta(A_1, 0))$ $(\because A_3 \subseteq A_2 \subseteq A_1)$

$= \in$-closure $(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0))$

$= \in$-closure $(\{q_0\} \cup \phi \cup \phi)$

$= A_1$

Similarly

$\hat{\delta}(A_1, 1) = A_2$

$\hat{\delta}(A_1, 2) = A_3$

$\hat{\delta}(A_2, 0) = \phi$

$\hat{\delta}(A_2, 1) = A_2$

$\hat{\delta}(A_2, 2) = A_3$

$\hat{\delta}(A_3, 0) = \phi$

$\hat{\delta}(A_3, 1) = \phi$

$\hat{\delta}(A_3, 2) = A_3$

Transition matrix is shown in table 1.30.

**Table 1.30**

| State \ Input | 0 | 1 | 2 |
|---|---|---|---|
| $q_0, q_1, q_2$ | $q_0, q_1, q_2$ | $q_1, q_2$ | $q_2$ |
| $q_1, q_2$ | ϕ | $q_1, q_2$ | $q_2$ |
| $q_2$ | ϕ | ϕ | $q_2$ |

So the required DFA is shown in fig. 1.43.

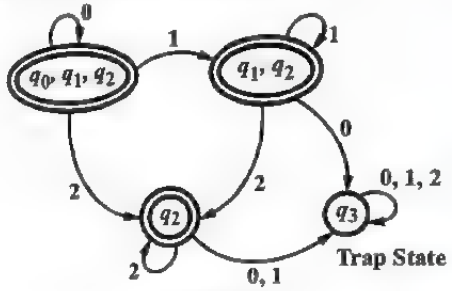



*Fig. 1.43*

***Prob.27. Draw a Mealy machine for the following language, outp string is identical to the input string on the even position.***

***(R.G.P.V., Dec. 20***

***Sol.*** The Mealy machine is given in fig. 1.44.

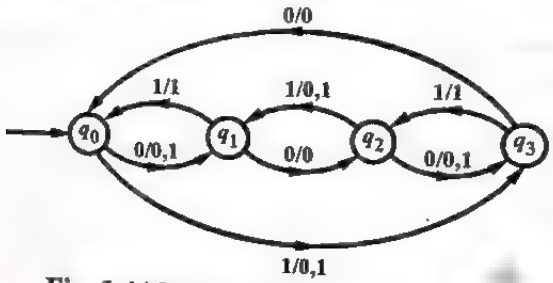



*Fig. 1.44 Mealy Machine for Prob.27*

Let us convert the transition diagram into transition table for the giv problem. Table 1.31 shows the required Mealy machine.

**Table 1.31 Mealy Machine for Problem**

| *Present State* | *Next State* | | | |
|---|---|---|---|---|
| | *a = 0* | | *a = 1* | |
| $q_0$ | $q_1$ | 0,1 | $q_3$ | 0,1 |
| $q_1$ | $q_2$ | 0 | $q_0$ | 1 |
| $q_2$ | $q_3$ | 0,1 | $q_1$ | 0,1 |
| $q_3$ | $q_0$ | 0 | $q_2$ | 1 |

***Prob.28. Construct a Moore machine which calculates mod-4 for eac binary string treated as binary integer.*** ***(R.G.P.V., Dec. 200***

***Sol.*** Since last two digits of every binary integer decides whether integ is divisible by 4 or not. So that it can also decide what will be the remainder the integer is divisible by 4 as



| *Last Two Digits of Binary Integer* | *Remainder* |
|---|---|
| 00 | 0 |
| 01 | 1 |
| 10 | 2 |
| 11 | 3 |

So, Moore machine which calculate mod-4 for each binary string treated as binary integer is shown in fig. 1.45.

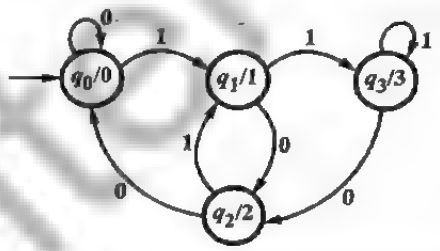



*Fig. 1.45 A Moore Machine to Calculate Mod-4 for Each Binary Integer*

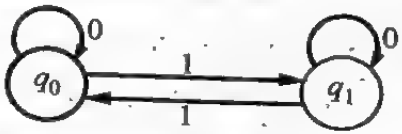
***Prob.29. Construct a Mealy machine which can output EVEN, ODD according as the total number of 1's encountered is even or odd. The input symbols are 0 or 1.*** ***(R.G.P.V., Dec. 2010)***

***Sol.*** We require two states $q_0$ and $q_1$. $q_0$ denotes even number of ones and $q_1$ denotes odd number of ones. First we construct F.S.M.

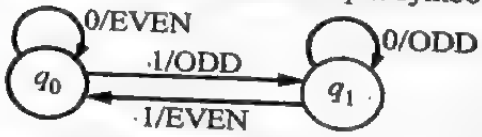
even no. of '1'
odd no. of '1'

| $Q$ \ $\Sigma$ | 0 | 1 |
|---|---|---|
| $q_0$ | $q_0$ | $q_1$ |
| $q_1$ | $q_1$ | $q_0$ |

Transition diagram for F.S.M.




Now, we will add the output with each input-symbol.




MEALY MACHINE

We can check that given string containing even numbers of ones or odd numbers of ones. Suppose input string is → 01110

$\underline{q_0}$ 1 1 1 0 ⇒ O/P → EVEN

0 $\underline{q_0}$ 1 1 1 0 ⇒ O/P → ODD

$0\ 1\ \underline{q_1}\ 1\ 1\ 0 \Rightarrow$ O/P → EVEN

$0\ 1\ 1\ \underline{q_0}\ 1\ 0 \Rightarrow$ O/P → ODD

$0\ 1\ 1\ 1\ \underline{q_0}\ 0 \Rightarrow$ O/P → ODD

$0\ 1\ 1\ 1\ 0\ \boxed{\underline{q_1}}$

$q_1$ shows that given string containing odd number of ones.

***Prob.30. Give Mealy and Moore machine for the input from (0 + 1)* the input ends in '000', output A; if the input ends in '111', output B otherwise output C.*** ***(R.G.P.V., Dec. 20***

***Sol.*** This machine is designed in such a way that if the input ends in 00 the output is *A*, if the input ends in 111, output is *B*. Otherwise output is *C*. partial design of it can be made as follows –

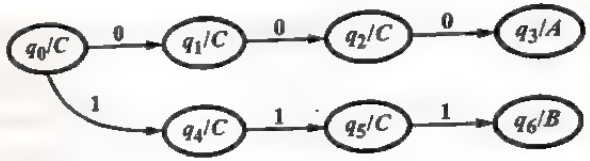



***Fig. 1.46***

Now, we will insert the possibilities of 1's and 0's for each state. Th the Moore machine becomes

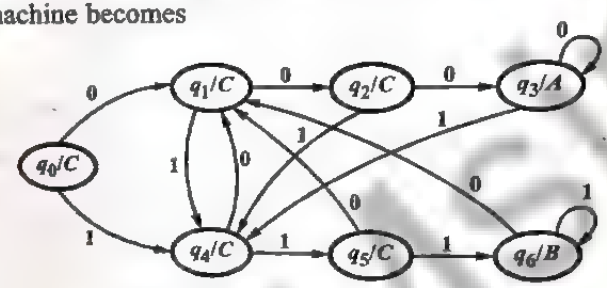



***Fig. 1.47 Moore Machine***

Now the Mealy machine will be as follows –

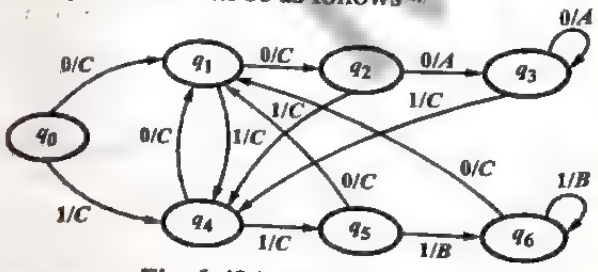



***Fig. 1.48 Mealy Machine***



***Prob.31. Construct the Mealy machine equivalent to the given Moore machine –***

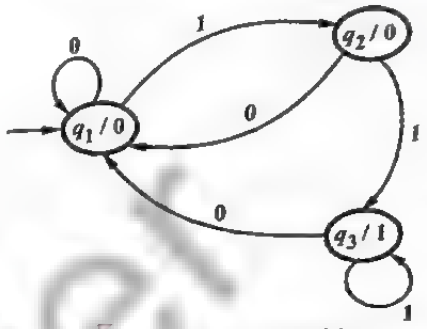



***Fig. 1.49 Moore Machine***

***Sol.*** First, we convert the transition diagram into the transition table as shown in table 1.32.

**Table 1.32 Transition Table for Moore Machine**

| *Present State* | *Next State* | | *Output* |
|---|---|---|---|
| | *a = 0* | *a = 1* | |
| → $q_1$ | $q_1$ | $q_2$ | 0 |
| $q_2$ | $q_1$ | $q_3$ | 0 |
| $q_3$ | $q_1$ | $q_3$ | 1 |

Now, we construct the transition table as shown in table 1.33 by associating the output with the transitions. The procedure is described in Q.17.

**Table 1.33 Intermediate Transition Table**

| *Present State* | *Next State* | | | |
|---|---|---|---|---|
| | *a = 0* | | *a = 1* | |
| | *State* | *Output* | *State* | *Output* |
| → $q_1$ | $q_1$ | 0 | $q_2$ | 0 |
| $q_2$ | $q_1$ | 0 | $q_3$ | 1 |
| $q_3$ | $q_1$ | 0 | $q_3$ | 1 |

In table 1.33, the rows corresponding to $q_2$ and $q_3$ are identical. So, we can delete one of the two states, i.e., $q_2$ or $q_3$. We reconstruct the table by deleting $q_3$ as shown in table 1.34.

**Table 1.34 Mealy Machine Equivalent to the Given Moore Machine**

| *Present State* | *Next State* | | | |
|---|---|---|---|---|
| | *a = 0* | | *a = 1* | |
| | *State* | *Output* | *State* | *Output* |
| → $q_1$ | $q_1$ | 0 | $q_2$ | 0 |
| $q_2$ | $q_1$ | 0 | $q_2$ | 1 |

The pair of states and outputs in the next state column can be rearrange as given in table 1.39.

***Prob.34. Construct Moore machine for the following Mealy machine***

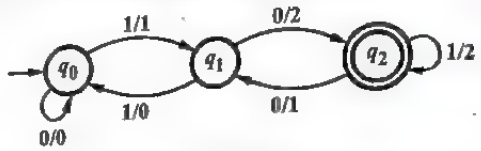



*Fig. 1.53*

***(R.G.P.V., June 20***

***Sol.*** Transition table of the fig. 1.53 is shown below –

**Table 1.40**

| State | Input | | | |
|---|---|---|---|---|
| | 0 | | 1 | |
| | State | Output | State | Output |
| $\rightarrow q_0$ | $q_0$ | 0 | $q_1$ | 1 |
| $q_1$ | $q_2$ | 2 | $q_0$ | 1 |
| $q_2$ | $q_1$ | 1 | $q_2$ | 2 |

Rearranged transition table is –

**Table 1.41**

| Present State | Next State | | Output |
|---|---|---|---|
| | $a = 0$ | $a = 1$ | |
| $q_{00}$ | $q_{00}$ | $q_1$ | 0 |
| $q_{01}$ | $q_{00}$ | $q_1$ | 1 |
| $q_1$ | $q_2$ | $q_{01}$ | 1 |
| $q_2$ | $q_1$ | $q_2$ | 2 |

Resultant Moore machine is shown in fig. 1.54.

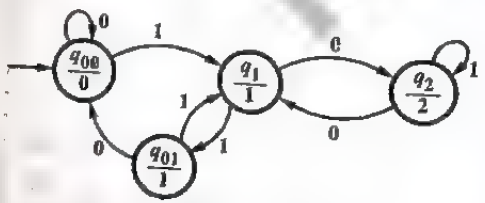



*Fig. 1.54*



***Prob.35. Convert the following Mealy machine into its equivalent Moore machine –***

| Present State | Next State | | | |
|---|---|---|---|---|
| | $a = 0$ | | $a = 1$ | |
| | State | Output | State | Output |
| $q_0$ | $q_1$ | $n$ | $q_2$ | $n$ |
| $q_1$ | $q_1$ | $y$ | $q_2$ | $n$ |
| $q_2$ | $q_1$ | $n$ | $q_2$ | $y$ |

***(R.G.P.V., Dec. 2012)***

***Sol.*** We construct the transition table as shown in table 1.42.

**Table 1.42 Intermediate State Table**

| Present State | Next State | | | |
|---|---|---|---|---|
| | $a = 0$ | | $a = 1$ | |
| | State | Output | State | Output |
| $q_0$ | $q_{1n}$ | $n$ | $q_{2n}$ | $n$ |
| $q_{1n}$ | $q_{1y}$ | $y$ | $q_{2n}$ | $n$ |
| $q_{1y}$ | $q_{1y}$ | $y$ | $q_{2n}$ | $n$ |
| $q_{2n}$ | $q_{1n}$ | $n$ | $q_{2y}$ | $y$ |
| $q_{2y}$ | $q_{1n}$ | $n$ | $q_{2y}$ | $y$ |

The pair of states and outputs in the next state column can be rearranged as given in table 1.43 and this table is equivalent to the Moore machine.

**Table 1.43 Moore Machine Equivalent to the Given Mealy Machine**

| Present State | Next State | | Output |
|---|---|---|---|
| | $a = 0$ | $a = 1$ | |
| $q_0$ | $q_{1n}$ | $q_{2n}$ | – |
| $q_{1n}$ | $q_{1y}$ | $q_{2n}$ | $n$ |
| $q_{1y}$ | $q_{1y}$ | $q_{2n}$ | $y$ |
| $q_{2n}$ | $q_{1n}$ | $q_{2y}$ | $n$ |
| $q_{2y}$ | $q_{1n}$ | $q_{2y}$ | $y$ |

The corresponding transition diagram is shown in fig. 1.55.

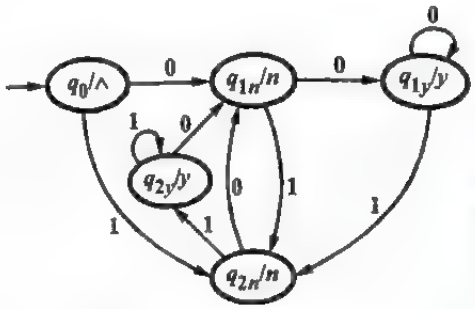



*Fig. 1.55 Moore Machine*

***Prob.36. Construct Moore machine equivalent to the given Mealy machine –***

**Table 1.44 Transition Table for Mealy Machine**

| State | Input | | | |
|---|---|---|---|---|
| | 0 | | 1 | |
| | State | Output | State | Output |
| $\rightarrow q_1$ | $q_1$ | 1 | $q_2$ | 0 |
| $q_2$ | $q_4$ | 1 | $q_4$ | 1 |
| $q_3$ | $q_2$ | 1 | $q_3$ | 1 |
| $q_4$ | $q_3$ | 0 | $q_1$ | 1 |

***(R.G.P.V., Dec. 2003, 20***

***Sol.*** The transition diagram corresponding to table 1.44 is shown fig. 1.56.

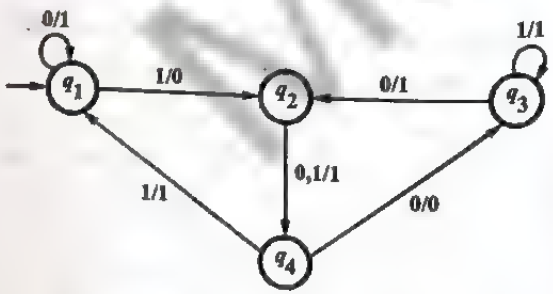



*Fig. 1.56 Mealy Machine*

Now, we construct the transition table as shown in table 1.45. T procedure is described in Q.16.



**Table 1.45 Intermediate State Table**

| Present State | Next State | | | |
|---|---|---|---|---|
| | a = 0 | | a = 1 | |
| | State | Output | State | Output |
| $\rightarrow q_1$ | $q_1$ | 1 | $q_{20}$ | 0 |
| $q_{20}$ | $q_4$ | 1 | $q_4$ | 1 |
| $q_{21}$ | $q_4$ | 1 | $q_4$ | 1 |
| $q_{30}$ | $q_{21}$ | 1 | $q_{31}$ | 1 |
| $q_{31}$ | $q_{21}$ | 1 | $q_{31}$ | 1 |
| $q_4$ | $q_{30}$ | 0 | $q_1$ | 1 |

The pair of states and outputs in the next state column can be rearranged as given in table 1.46.

**Table 1.46 Rearranged State Table**

| Present State | Next State | | Output |
|---|---|---|---|
| | a = 0 | a = 1 | |
| $\rightarrow q_1$ | $q_1$ | $q_{20}$ | 1 |
| $q_{20}$ | $q_4$ | $q_4$ | 0 |
| $q_{21}$ | $q_4$ | $q_4$ | 1 |
| $q_{30}$ | $q_{21}$ | $q_{31}$ | 0 |
| $q_{31}$ | $q_{21}$ | $q_{31}$ | 1 |
| $q_4$ | $q_{30}$ | $q_1$ | 1 |

Since, the initial state $q_1$ is associated with output 1. This means that with input $\Lambda$ we get an output of 1, if the machine starts at state $q_1$. To overcome this situation, we add a new starting state $q_0$, whose state transitions are identical with those of $q_1$ but whose output is 0. So, table 1.45 is transformed to table as shown in table 1.47.

**Table 1.47 Moore Machine Equivalent to the Given Mealy Machine**

| Present State | Next State | | Output |
|---|---|---|---|
| | a = 0 | a = 1 | |
| $\rightarrow q_0$ | $q_1$ | $q_{20}$ | 0 |
| $q_1$ | $q_1$ | $q_{20}$ | 1 |
| $q_{20}$ | $q_4$ | $q_4$ | 0 |
| $q_{21}$ | $q_4$ | $q_4$ | 1 |
| $q_{30}$ | $q_{21}$ | $q_{31}$ | 0 |
| $q_{31}$ | $q_{21}$ | $q_{31}$ | 1 |
| $q_4$ | $q_{30}$ | $q_1$ | 1 |

The corresponding transition diagram is shown in fig. 1.57.

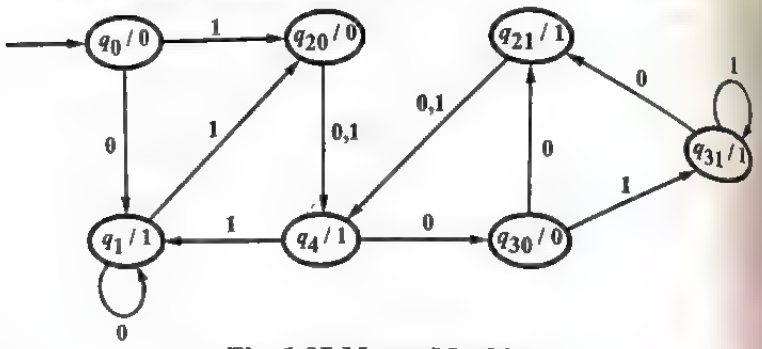



*Fig. 1.57 Moore Machine*

# UNIT 2

## REGULAR GRAMMARS, REGULAR EXPRESSIONS, REGULAR SETS, CLOSURE PROPERTIES OF REGULAR GRAMMARS, ARDEN'S THEOREM, MYHILL-NERODE THEOREM, PUMPING LEMMA FOR REGULAR LANGUAGES, APPLICATION OF PUMPING LEMMA

***Q.1. What do you mean by alphabets ?***

***Ans.*** An alphabet is a finite set of symbols. In the case of a common language such as English, we would want the alphabet to include 26 letters, both uppercase and lowercase, as well as blanks and various punctuation symbols. In the case of a programming language, we would want to add the 10 numeric digits.

***Q.2. What do you mean by a string ?***

***Ans.*** A string (or word) is a finite sequence of symbols juxtaposed. For example, $a$, $b$, and $c$ are symbols and $abcb$ is a string.

The ***length*** of a string $w$, denoted $|w|$, is the number of symbols in the string. For example, $abcb$ has length 4. The empty string denoted by $\epsilon$, is the string consisting of zero symbols.

Thus, $$|\epsilon| = 0$$

A ***prefix*** of a string is any number of leading symbols of that string and a ***suffix*** is any number of trailing symbols. For example, string $abc$ has prefixes $\epsilon$, $a$, $ab$ and $abc$, and its suffixes are $\epsilon$, $c$, $bc$ and $abc$. A prefix or suffix of a string other than the string itself, is called a proper prefix or suffix.

The ***concatenation*** of two strings $w$ and $v$ is the string obtained by appending the symbols of $v$ to the right end of $w$, that is, if

$$w = a_1a_2 \ldots\ldots a_n$$

and

$$v = b_1b_2 \ldots\ldots b_m$$

then the concatenation of $w$ and $v$, denoted by $wv$, is

$$wv = a_1a_2 \ldots\ldots a_n\ b_1b_2\ldots\ldots b_m$$

The empty string is the identity for the concatenation operator. That is, $\in w = w \in = w$ for each string $w$. The ***reverse*** of a string is obtained by writing the symbols in reverse order, if $w$ is a string as shown above, then its reverse is

$$w^R = a_n \ldots a_2 a_1$$

***Q.3. Explain the term alphabet and string.*** ***(R.G.P.V., June***

***Ans.*** Refer to Q.1 and Q.2.

***Q.4. What do you understand by a regular grammar ?***

***Or***

***Explain the left linear grammar.*** ***(R.G.P.V., Dec. 2***

***Ans.*** A ***regular grammar*** is one that is either left-linear or right-linear.

A grammar $G = (V, T, P, S)$ is said to be ***left-linear*** if all productions are of the form –

$$A \to Bx \quad \text{or} \quad A \to x$$

where $A, B, \in V$, and $x \in T^*$

A grammar is said to be ***right-linear*** if all productions are of the form

$$A \to xB$$
$$A \to x$$

Note that, in a regular grammar at most one variable appears on the right side of any production. Moreover, that variable must consistently be either the leftmost or rightmost symbol of the right side of any production.

***Q.5. How can we construct regular grammar from regular expressi***

***(R.G.P.V., June 2***

***Ans.*** We can construct regular grammar from regular expression by using following method –

(i) Construct a NFA with $\in$ from given regular expression.

(ii) Eliminate $\in$ transitions and convert it to equivalent DFA.

(iii) From constructed DFA, the corresponding states become non-terminal symbols and transitions made are equivalent to production rules.

***Q.6. Describe regular expressions. Also, give the formal definition of regular expression.***

***Or***

***Write short note on regular expressions. (R.G.P.V., Dec. 2007, June 20***

***Or***

***Define regular expressions.***

***(R.G.P.V., Dec. 20***

***Ans.*** The languages accepted by finite automata are easily described by simple expressions called regular expressions. Regular expressions are used for representing certain sets of strings in an algebraic fashion. This notation for regular expression involves a combination of strings of symbols from some alphabet over $\Sigma$, parenthesis, and the operators such as + and *.



**Formal Definition of a Regular Expression** – Regular expressions are constructed from primitive constituents by repeatedly applying certain recursive rules. This is much similar to the way the familiar arithmetic expressions are constructed. The regular expressions over $\Sigma$ and the sets that they denote are defined recursively as follows –

(i) Any terminal symbol (i.e., an element of $\Sigma$), $\Lambda$, $\phi$, are regular expressions. These are called ***primitive regular expressions***

(ii) The union of two regular expressions $R_1$ and $R_2$, written as $R_1 + R_2$, is also a regular expression

(iii) The concatenation of two regular expressions $R_1$ and $R_2$, written as $R_1R_2$, is also a regular expression

(iv) The iteration (or closure) of a regular expression $R$, written as $R^*$, is also a regular expression

(v) If $R$ is a regular expression, then $(R)$ is also a regular expression.

In the absence of parenthesis, we have the hierarchy of operations as follows – iteration (closure), concatenation, and union. i.e., in evaluating a regular expression involving various operations, we perform iteration first, then concatenation, and finally union.

***Q.7. Write the identities of regular expression.*** ***(R.G.P.V., Dec. 2017)***

***Ans.*** Two regular expressions, say $P$ and $Q$, are equivalent (i.e., $P = Q$) if $P$ and $Q$ represent the same set of strings.

Following are the identities for regular expressions. These are useful for simplifying regular expressions –

(i) $\phi + R = R$
(ii) $\phi R = R\phi = \phi$
(iii) $\Lambda R = R\Lambda = R$
(iv) $\Lambda^* = \Lambda$ and $\phi^* = \Lambda$
(v) $R + R = R$
(vi) $R^*R^* = R^*$
(vii) $RR^* = R^*R$
(viii) $(R^*)^* = R^*$
(ix) $\Lambda + RR^* = R^* = \Lambda + R^*R$
(x) $(PQ)^* P = P(QP)^*$
(xi) $(P + Q)^* = (P^*Q^*)^* = (P^* + Q^*)^*$
(xii) $(P + Q)\,R = PR + QR$ and $R(P + Q) = RP + RQ$.

('Set $P$' means that the set represented by the regular expression $P$)

***Q.8. Define the language associated with regular expressions.***

***Ans.*** Regular expressions can be used to describe some simple languages. If $R$ is a regular expression, we will let $L(R)$ denote the language associated

with $R$. This language is defined as follows –

The language $L(R)$ denoted by any regular expression $R$ is defined by following rules –

(i) $\phi$ is a regular expression, denoting the empty set

(ii) $\Lambda$ is a regular expression, denoting $\{\Lambda\}$

(iii) For every $a \in \Sigma$, $a$ is a regular expression denoting $\{a\}$.

If $R_1$ and $R_2$ are regular expressions, then –

(iv) $L(R_1 + R_2) = L(R_1) \cup L(R_2)$ (v) $L(R_1 R_2) = L(R_1)$

(vi) $L((R_1)) = L(R_1)$ (vii) $L(R_1^*) = (L(R_1))^*$

The last four rules are used to reduce $L(R)$ to simpler compo recursively. While the first three are the termination conditions fo recursion.

For example, we can exhibit the language $L(a^*(a + b))$ in set not follows –

$$\begin{aligned} L(a^*(a+b)) &= L(a^*)\, L(a+b) \\ &= (L(a))^*\, (L(a) \cup L(b)) \\ &= \{\Lambda, a, aa, \ldots\ldots\}\ \{a, b\} \\ &= \{a, aa, aaa \ldots\ldots\ldots, b, ab, aab, \ldots\ldots \end{aligned}$$

But, there is a problem with rules (iv) to (vii) in the definition define a language precisely if $R_1$ and $R_2$ are given, but there may be ambiguity in dividing a complicated expression into parts. For example, the regular expression $ab + c$, made up of $R_1 = ab$ and $R_2 = c$. In this cas find $L(ab + c) = \{ab, c\}$. But, there is nothing in the definition to stop us taking $R_1 = a$ and $R_2 = b + c$. Now, we get a different result, $L(ab + c)$ $ac\}$. To get overcome this, we require that all expressions be fully parenthe but this makes the result cumbersome. Thus, we use a convention fam from mathematics and programming languages. We can make a st precedence rules for evaluation in which star-closure (*) precedes concate and concatenation precedes union.

***Q.9. Define regular set. Explain it with examples.***

***Or***

***Write short note on regular languages.*** ***(R.G.P.V., June 2005,***

***Ans.*** A regular set is the set represented by a regular expression. For exa in $a, b \in \Sigma$, then $a$ denotes the set $\{a\}$, $a + b$ denotes $\{a, b\}$, $ab$ denotes $a^*$ denotes the set $\{\Lambda, a, aa, aaa, \ldots\ldots\}$ and $(a + b)^*$ denotes $\{a, b\}^*$.

For example, the following sets can be described as regular expres

(i) $\{1\}$, $\{0\}$ are represented by **1** and **0**, respectively. obtained by concatenating 1, 0 and 1. So, $\{101\}$ is represented by **101**

(ii) $\{abba\}$ is represented by ***abba***



(iii) As $\{ab, ba\}$ is the union of $\{ab\}$ and $\{ba\}$, $\{ab, ba\}$ is repre- nted by $ab + ba$

(iv) The set $\{\Lambda, 10\}$ is represented by $\Lambda + 10$

(v) The set $\{abb, a, b, bba\}$ is represented by $abb + a + b + bba$

(vi) As $\{\Lambda, a, aa, aaa, \ldots\ldots\}$ is simply $\{a\}^*$, it is represented by $a^*$

(vii) Any element in $\{1, 11, 111, \ldots\}$ can be obtained by oncatenating 1 and any element of $\{1\}^*$. Hence $1(1)^*$ represents $\{1, 11, 111, ....\}$. It is denoted as $\{1\}^+$.

***Q.10. How can you test the equivalence of two regular expressions ?***

***Ans.*** Let we have two regular expressions $P$ and $Q$. $P$ and $Q$ are equivalent and only if they represent the same set. Also, they are equivalent if and only the corresponding finite automata are equivalent.

The equivalence of $P$ and $Q$ can be proves by three ways –

(i) Prove that the sets $P$ and $Q$ are the same. (For non-equivalence, nd a string in one set but not in the other).

(ii) Use the identities to prove the equivalence of $P$ and $Q$

(iii) Construct the corresponding finite automata $M$ and $M'$ and prove at they are equivalent. (For non-equivalence, prove that $M$ and $M'$ are not quivalent)

One of the above three methods can be chosen according to the problem.

***Q.11. State and prove the theorem describing the relation between ransition systems and regular expressions.***

***Ans.*** The languages accepted by finite automata are precisely the languages enoted by regular expressions. So that there is an NFA with $\in$-transitions enoting the same language.

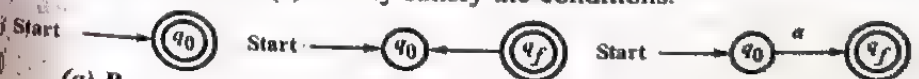
The following theorem describes the relation between transition systems nd regular expressions –

**Theorem.** Let $R$ be a regular expression. Then there exists an NFA with -transitions that accepts $L(R)$.

**Proof.** We show by induction on the number of operators in the regular xpression $R$ that there is an NFA with $\in$-transitions, having one final state and o transitions out of this final state, such that

$$L(M) = L(R)$$

**Basis** – The expression $R$ must be $\in$, $\phi$, or $a$ for some $a$ in $\Sigma$. The NFAs n fig. 2.1 (a), (b) and (c) clearly satisfy the conditions.




*(a) $R = \in$* *(b) $R = \phi$* *(c) $R = a$*

***Fig. 2.1 Transition Systems Recognizing Elementary Regular Sets***

**Induction Step** – Assume that the theorem is true for regular expr with fewer than $i$ operators, $i \geq 1$. Let $R$ have $i$ operators. There ar cases depending on the form of $R$.

*(i) Case 1* – For union, i.e., $R = R_1 + R_2$. Both $R_1$ and $R_2$ have fewer than $i$ operators. Thus there are NFAs $M_1 = (Q_1, \Sigma_1, \delta_1, q_1$ and $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, \{f_2\})$ with $L(M_1) = L(R_1)$ and $L(M_2) =$ Since, we may rename states of an NFA at will we may assume $Q_1$ and disjoint. Let $q_0$ be a new initial state and $f_0$ a new final state. Then we co

$$M = (Q_1 \cup Q_2 \cup \{q_0, f_0\}, \Sigma_1 \cup \Sigma_2, \delta, q_0, \{f_0\}$$

where $\delta$ is defined by,

(a) $\delta(q_0, \in) = \{q_1, q_2\}$

(b) $\delta(q, a) = \delta_1(q, a)$ for $q$ is $Q_1 - \{f_1\}$ and $a$ in $\Sigma_1 \cup$

(c) $\delta(q, a) = \delta_2(q, a)$ for $q$ is $Q_2 - \{f_2\}$ and $a$ in $\Sigma_2 \cup$

(d) $\delta(f_1, \in) = \delta_1(f_2, \in) = \{f_0\}$.

Recall by the inductive hypothesis that there are no transitions $f_1$ or $f_2$ in $M_1$ or $M_2$. Thus all the moves of $M_1$ and $M_2$ are present in $M$ construction of $M$ is depicted in fig. 2.2. It follows that there is labelled $x$ in $M$ from $q_0$ to $f_0$ if and only if there is a path labelled $x$ from $q_1$ to $f_1$ or a path in $M_2$ from $q_2$ to $f_2$. Hence $L(M) = L(M_1) \cup$ as desired.

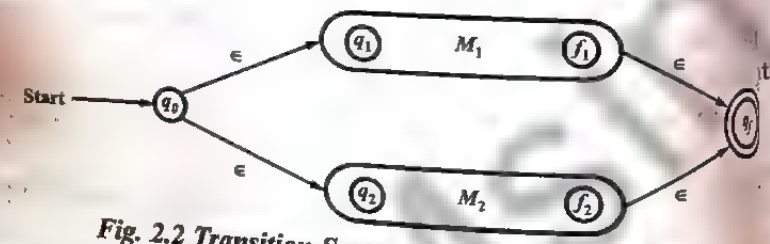



*Fig. 2.2 Transition System Recognizing $R = R_1 + R_2$*

*(ii) Case 2* – For concatenation i.e., $R = R_1R_2$. Let $M_1$ and $M_2$ in case 1 and then we construct

$$M = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta, \{q_1\}, \{f_2\})$$

where $\delta$ is given by,

(a) $\delta(q, a) = \delta_1(q, a)$ for $q$ in $Q_1 - \{f_1\}$ and $a$ in $\Sigma_1 \cup$

(b) $\delta(f_1, \in) = q_2$

(c) $\delta(q, a) = \delta_2(q, a)$ for $q$ in $Q_2$ and $a$ in $\Sigma_2 \cup \{\in\}$

The construction of $M$ is depicted in fig. 2.3. Every path in $M$ from $q_1$ to $f_2$ is a path labelled by some string $x$ from $q_1$ to $f_1$, followed by the edge $f_1$ to $q_2$ labelled $\in$, followed by a path labelled by some string $y$ from $q_2$ to $f_2$. Thus $L(M) = L(M_1)\, L(M_2)$ as desired.



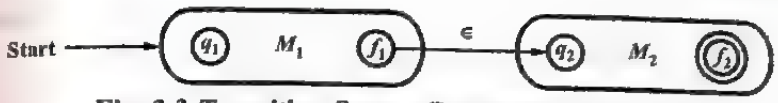



*Fig. 2.3 Transition System Recognizing $R = R_1R_2$*

*(iii) Case 3* – For closure, i.e., $R = R_1^*$. Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1,$ and $L(M_1) = R_1$. Then we construct

$$M = (Q_1 \cup \{q_0, f_0\}, \Sigma_1, \delta, q_0, \{f_0\}$$

here $\delta$ is given by –

(a) $\delta(q_0, \in) = \delta(f_1, \in) = \{q_1, f_0\}$

(b) $\delta(q, a) = \delta_1(q, a)$ for $q$ in $Q_1 - \{f_1\}$ and $a$ in $\Sigma_1 \cup \{\in\}$

The construction of $M$ is depicted in fig. 2.4. It follows that there is a ath in $M$ from $q_0$ to $f_0$ labelled $x$ if and only if we can write $x = x_1, x_2 \ldots x_j$ for me $j \geq 0$ (the case $j = 0$ means $x = \in$) such that each $x_i$ is in $L(M_1)$ $L_1(M_1)^*$ as desired.

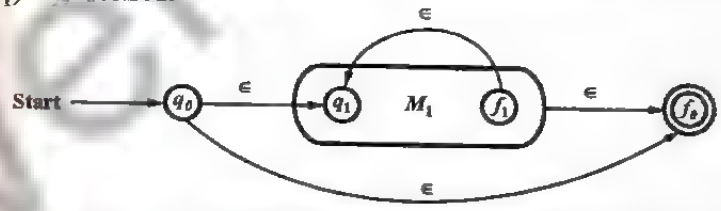



*Fig. 2.4 Transition System Recognizing $R = R_1^*$*

This proof is in essence an algorithm for converting a regular expression to an equivalent finite automaton.

***Q.12. Explain the closure properties of regular sets.***

***Or***

***Prove that the regular sets are closed under intersection.***

***(R.G.P.V., June 2003, Dec. 2003, June 2004)***

***Or***

***Show that regular languages are closed under complement and abtraction operation.*** ***(R.G.P.V., Dec. 2002)***

***Or***

***Write short note on closure property of regular grammar.***

***(R.G.P.V., Dec. 2017)***

***Or***

***Mention the closure properties of regular languages. (R.G.P.V., Nov. 2018)***

***Ans.*** If $L_1$ and $L_2$ are regular languages, then there exist regular expression and $R_2$ such that $L_1 = L(R_1)$ and $L_2 = L(R_2)$. By definition, $+ R_2$, $R_1R_2$, and $R_1^*$ are regular expressions denoting the languages $\cup L_2$, $L_1L_2$, and $L_1^*$, respectively. Thus, closure under union, oncatenation, and star-closure is immediate.

Following theorems prove the other closure properties of regular langu

**Theorem 1.** If $L$ is regular then transpose $L^T$ is also regular.

**Proof.** Every regular expression is recognized by transition syst transition system $M$ can be converted into finite accepter (FA), accepti same set as $M$. Any set accepted by FA is represented by a regular expr A subset $L$ of $\Sigma^*$ is regular set if and only if it is accepted by $FA$, so $L$ is re We can construct a $FA$ $M = (Q, \Sigma, \delta, q_0, F)$ such that

$$T(M) = L$$

We construct a transition system $M'$ by starting with the state diag $M$ and reversing the direction of the directed edges. The set of initial sta $M'$ is defined as the set $F$, and $q_0$ is defined as the (only) final state of $M'$ $M' = (Q, \Sigma, \delta', F, \{q_0\})$.

If $w \in T(M)$, we have a path from $q_0$, to some final state in $F$ with value $w$. By 'reversing the edges', we get a pair in $M'$ from some final st $F$ to $q_0$. Its path value is $w^T$. So $w^T \in T(M')$. In a similar way, we can sa if $w_1 \in T(M')$, then ${w_1}^T \in T(M)$. Thus from the state diagram, it is easy to that $T(M') = T(M)^T$. We can prove rigorously that $w \in T(M)$ if and only if $T(M')$ by induction on $|w|$. So $T(M)^T = T(M')$ we know that, a $M$ on $h(a)$. Formally, let $M' = (Q, \Delta, \delta', q_0, F)$ and define $\delta'(q, a)$, for $q$ in $Q$ $L$ of $\Sigma^*$ is a regular set if and only if it is recognized by a transition system and $a$ in $\Delta$ to be $\delta(q, h(a))$. Note that $h(a)$ may be a long string, or $\in$, but $\delta$ is $T(M')$ is regular, i.e., $T(M)^T$ is regular.

**Theorem 2.** If $L$ is a regular set over $\Sigma$, then the complement of $L$ $\Sigma^* - L$ is also regular over $\Sigma$.

**Proof.** As $L$ is regular if it is accepted by FA. We can construct a $M = (Q, \Sigma, \delta, q_0, F)$ accepting $L$, i.e., $L = T(M)$.

We construct another DFA $M' = (Q, \Sigma, \delta, q_0, F')$ by defining $F' = Q$ i.e., $M$ and $M'$ differ only in their final states. A final state of $M'$ is a non state of $M$ and vice versa. The state diagrams of $M$ and $M'$ are the except for the final states.

$w \in T(M')$ if and only if $\delta(q_0, w) \in F' = Q - F$, i.e., if and o $w \notin L$. This proves

$$T(M') = \Sigma^* - L.$$

**Theorem 3.** If $L_1$ and $L_2$ are regular sets over $\Sigma_1$ then $L_1 \cap L_2$ is regular over $\Sigma$.

**Proof.** For closure under intersection, we start with DeMorgan's l

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

for any languages $L_1$ and $L_2$. Now, if $L_1$ and $L_2$ are regular, then by cl under complementation, so are $\overline{L_1}$ and $\overline{L_2}$. Using closure under union next get that $\overline{L_1} \cup \overline{L_2}$ is regular. Using closure under complementation more, we see that $\overline{\overline{L_1} \cup \overline{L_2}}$ is regular, i.e., $L_1 \cap L_2$ is regular.



**Theorem 4.** The class of regular sets is closed under substitution.

**Proof.** Let $R \subseteq \Sigma^*$ be a regular set and for each $a$ in $\Sigma$ let $R_a \subseteq \Delta^*$ be regular set. Let $f : \Sigma \to \Delta^*$ be the substitution defined by $f(a) = R_a$. Select gular expressions denoting $R$ and each $R_a$. Replace each occurrence of the mbol $a$ in the regular expression for $R$ by the regular expression for $R_a$. To ove that the resulting regular expression denotes $f(R)$, observe that the bstitution of a union, product, or closure is the union, product or closure of e substitution. A simple induction on the number of operators in the regular pression completes the proof.

**Theorem 5.** The class of regular sets is closed under homomorphisms nd inverse homomorphisms.

**Proof.** Closure under homomorphisms follows immediately from closure nder substitution, since every homomorphism is a substitution, in which $(a)$ has one member.

To show closure under inverse homomorphism, let $M = (Q, \Sigma, \delta, q_0, F)$ e a DFA accepting $L$, and let $h$ be a homomorphism from $\Delta$ to $\Sigma^*$. We onstruct a DFA $M'$ that accepts $h^{-1}(L)$ by reading symbol $a$ in $\Delta$ and simulating on $h(a)$. Formally, let $M' = (Q, \Delta, \delta', q_0, F)$ and define $\delta'(q, a)$, for $q$ in $Q$ and $a$ in $\Delta$ to be $\delta(q, h(a))$. Note that $h(a)$ may be a long string, or $\in$, but $\delta$ is efined on all strings by extension. It is easy to show by induction on $|x|$ that $'(q_0, x) = \delta(q_0, h(x))$. Therefore $M'$ accepts $x$ if and only if $M$ accepts $h(x)$. hat is, $L(M') = h^{-1}(L(M))$.

**Theorem 6.** The class of regular sets is closed under quotient with arbitrary sets.

**Proof.** Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton accepting some egular set R and let L be an arbitrary language. The quotient $R/L$ is accepted y a finite automaton $M' = (Q, \Sigma, \delta, q_0, F')$, which behaves like $M$ except that he final states of $M'$ are all states $q$ of $M$ such that there exists $y$ in $L$ for which $\delta(q, y)$ is in $F$. Then $\delta(q_0, x)$ is in $F'$ if and only if there exists $y$ such hat $\delta(q_0, xy)$ is in $F$. Thus, $M'$ accepts $R/L$.

***Q.13. State and prove Arden's theorem.***

***Ans.*** It is very much useful in simplifying regular expressions.

It can be given as follows –

Let $P$ and $Q$ be two regular expressions over $\Sigma$. If $P$ does not contain $\Lambda$, hen the following equation –

$$R = Q + RP \quad \text{....(i)}$$

as a unique solution (i.e., one and only one solution) given by –

$$R + QP^*$$

**Proof –**

$$Q + (QP^*)\,P = Q\,(\Lambda + P^*P)$$

$$= QP^* \qquad \text{(by identity (ix) of Q.7)}$$

Hence equation (i) is satisfied when $R = QP^*$. It means that $R = QP^*$ is a solution of equation (i).

Now, to prove uniqueness, take equation (i). Here, we replace $Q + RP$ on the R.H.S., we get

$$\begin{aligned} Q + RP &= Q + (Q + RP)\, P \\ &= Q + QP + RPP \\ &= Q + QP + RP^2 \\ &= Q + QP + QP^2 + \ldots\ldots\ldots\ldots + QP^i + RP^{i+1} \\ &= Q\,(\Lambda + P + P^2 + \ldots\ldots\ldots\ldots P^i) + RP^{i+1} \end{aligned}$$

From equation (i),

$$R = Q\,(\Lambda + P + P^2 + \ldots + P^i) + RP^{i+1} \quad \text{for } i \geq 0$$

Now we show that any solution of equation (i) is equivalent to $QP^*$. Suppose $R$ satisfies equation (i), then it satisfies equation (ii). Let $u$ be of length $i$ in the set $R$. Then $u$ belongs to the set $Q\,(\Lambda + P + P^2 + \ldots + P^i) + RP^{i+1}$. As $P$ does not have $\Lambda$, $RP^{i+1}$ has no string of length less $i+1$ and so $u$ is not in the set $RP^{i+1}$. This means $u$ belongs to $Q(\Lambda + P + P^2 + \ldots\ldots\ldots\ldots + P^i)$, and hence to $QP^*$.

Consider a string $u$ in the set $QP^*$. Then $u$ is in the set $QP^k$ for $k \geq 0$ and hence in $Q(\Lambda + P + P^2 + \ldots\ldots\ldots\ldots + P^k)$. So $u$ is on the R.H. equation (ii). Therefore $u$ is in $R$ [L.H.S. of equation (ii)]. Thus $R$ and represent the same set. This proves the uniqueness of the solution of equati

***Q.14. Explain Myhill-Nerode theorem.*** ***(R.G.P.V., June***

***Or***

***State and prove Myhill-Nerode theorem.***

***(R.G.P.V., June 2003, Dec. 2007,***

***Or***

***Write short note on Myhill-Nerode theorem.*** ***(R.G.P.V., June***

***Or***

***Write the Myhill-Nerode theorem.*** ***(R.G.P.V., Dec.***

***Or***

***State the Myhill-Nerode theorem.*** ***(R.G.P.V., June***

***Or***

***Write and explain Myhill-Nerode theorem.*** ***(R.G.P.V., June***

***Or***

***Explain Myhill-Nerode theorem with example.*** ***(R.G.P.V., Dec.***

**Ans.** An equivalence relation $R$ such that $xRy$ implies $xzRyz$ is said ***right invarient*** (with respect to concatenation). We see that every finite autom induces a right invarient equivalence relation, defined as $R_m$ was defined set of input strings. This result is formalized in the following theorem.



**Theorem.** The following three statements are equivalent –

(i) The set $L \subseteq \Sigma^*$ is accepted by some finite automaton

(ii) $L$ is the union of some of the equivalence classes of a right varient equivalence relation of finite index

(iii) Let equivalence relation $R_L$ be defined by – $xR_Ly$ if and only if r all $z$ in $\Sigma^*$, $xz$ is in $L$ exactly when $yz$ is in $L$. Then $R_L$ is of finite index.

**Proof.** (i) → (ii) Assume that $L$ is accepted by some DFA $M = (Q, \Sigma, \delta, F)$. Let $R_m$ be the equivalence relation $xR_my$ if and only if $\delta(q_0, x) = \delta(q_0, y)$. $R_m$ is right invarient since, for any $z$, if $\delta(q_0, x) = \delta(q_0, y)$, then $\delta(q_0, xz) = \delta(q_0, yz)$. The index of $R_m$ is finite, since the index is at most the number of states in $Q$. Moreover, $L$ is the union of those equivalence classes that include a string $x$ such that $\delta(q_0, x)$ is in $F$, i.e., the equivalent classes corresponding to final states.

(ii) → (iii) We show that any equivalence relation $E$ satisfying (ii) is a refinement of $R_L$, i.e., every equivalence class of $E$ is entirely contained in some equivalence class of $R_L$. Thus, the index of $R_L$ cannot be greater than the index of $E$ and so is finite. Suppose that $xEy$. Then as $E$ is the right varient, for each $z$ in $\Sigma^*$, $xzEyz$, and thus $yz$ is in $L$ if and only if $xz$ is in $L$. Thus $xR_Ly$, and hence the equivalence class of $x$ in $R_L$. Now, we conclude that each equivalence class of $E$ is contained within some equivalence class of $R_L$.

(iii) → (i) First, we must show that $R_L$ is right invarient. Suppose $xR_Ly$, and let $u$ be in $\Sigma^*$. We have to prove that $xuR_Lyu$, i.e., for any $z$, $xuz$ is in $L$ exactly when $yuz$ is in $L$. But since $xR_Ly$, by the definition of $R_L$ we know that for any $v$, $xv$ is in $L$ exactly when $yv$ is in $L$. Let $v = wz$ to prove that $R_L$ is right invarient.

Now let $Q'$ be the finite set of equivalence classes of $R_L$ and $[x]$ the element of $Q'$ containing $x$. Define $\delta'([x], a) = [xa]$. The definition is consistent, since $R_L$ is right invarient. Had we chosen $y$ instead of $x$ from the equivalence class $[x]$, we would have obtained $\delta'([x], a) = [ya]$. But $xR_Ly$, so $xz$ is in $L$ exactly, when $yz$ is in $L$. Particularly, if $z = az'$, $xaz'$ is in $L$ exactly when $yaz'$ is in $L$, so $xaR_Lya$, and $[xa] = [ya]$. Let $q_0' = [\in]$ and let $F' = \{[x] \mid x \text{ is in } L\}$. The finite automaton $M' = (Q', \Sigma, \delta', q_0', F')$ accepts $L$, since $\delta'(q_0, x) = [x]$, and thus $x$ is in $L(M')$ if and only if $[x]$ is in $F'$.

***Q.15. Give the minimization algorithm to minimize the finite automata with the help of Myhill-Nerode theorem.***

***Ans.*** The Myhill-Nerode theorem has the implication that there is an essentially unique minimum state DFA for every regular set.

**Theorem** – The minimum state automaton accepting a set $L$ is unique upto an isomorphism (i.e., renaming of the states) and is given by $M'$.

**Proof** – In the proof of Myhill-Nerode theorem, we saw that an $M = (Q, \Sigma, \delta, q_0, F)$ accepting $L$ defines an equivalence relation $R_M$ refinement of $R_L$. Therefore, the number of states of $M$ is greater than equal to the number of states of $M'$. If equality holds, then each of the of $M$ can be identified with one of the states of $M'$. i.e., let $q$ be a state There must be some $x$ in $\Sigma^*$, such that $\delta(q_0, x) = q$, otherwise $q$ co removed from $Q$, and a smaller automation found. Identify $q$ with $\delta'(q_0', x)$, of $M'$. This identification will be consistent. If $\delta(q_0, x) = q$, $= q$, then, by the proof of Myhill-Nerode theorem, $x$ and $y$ are in the equivalence class of $R_L$. Thus, $\delta'(q_0', x) = \delta'(q_0', y)$.

**Minimization Algorithm** – There is a simple method for finding minimum state DFA $M'$ equivalent to a given DFA $M = (Q, \Sigma, \delta, q_0, F)$ $\equiv$ be the equivalence relation on the states of $M$ such that $p \equiv q$ if and for each input string $x$, $\delta(p, x)$ is an accepting state if and only if $\delta(q, x)$ accepting state. It is observed that there is an isomorphism between equivalence classes of $\equiv$ that contain a state reachable from $q_0$ by some string and the states of the minimum state FA $M'$. Thus the states of $M'$ be identified with these classes.

Rather than give a formal algorithm for computing the equivalence of $\equiv$ we first work through an example first some terminology is need $\equiv q$, we say $p$ is equivalent to $q$. We say that $p$ is distinguishable from $q$ exists an $x$ such that $\delta(p, x)$ is in $F$ and $\delta(q, x)$ is not, or vice-versa.

***Q.16. Explain Myhill-Nerode method of minimization.***
***(R.G.P.V., June 2009, Nov***

***Ans.*** Refer to Q.15.

***Q.17. State and prove pumping lemma for regular sets.***
***(R.G.P.V., June***

***Or***

***State and prove the pumping lemma theory of regular languag***
***(R.G.P.V., June 2009, Nov***

***Or***

***Write short note on pumping lemma for regular language.***
***(R.G.P.V., Dec.***

***Ans. Pumping lemma*** gives a method of pumping (or generating) input strings from a given string. It also gives a necessary condition input string to belong to a regular set. Hence, it can be used to show certain sets are not regular. It can be given as follows –

**Pumping Lemma** – Let $M(Q, \Sigma, \delta, q_0, F)$ be a finite automaton states. Let $L$ be the regular set accepted by $M$. Let $u \in L$ and $|u| \geq n$. If



there exists $x, y, z$ such that –

$u = xyz, y \neq \Lambda$, and $xy^iz \in L$, for each $i \geq 0$.

**Proof** – Let $u = a_1a_2a_3 \ldots a_m$, $m \geq n$

$\delta(q_0, a_1a_2a_3 \ldots a_i) = q_i$, for $i = 1, 2, \ldots, m$

$Q_1 = \{q_0, q_1, q_2 \ldots\ldots, q_m\}$

e., $Q_1$ is the sequence of states in the path with path value $u = a_1a_2a_3 \ldots a_m$. ince there are only $n$ distinct states, at least two states in $Q_1$ must coincide. mong various repeated states pairs, take the first pair. Let us take them as $q_j$ nd $q_k$ ($q_j = q_k$). Then $j$ and $k$ satisfy the condition $0 \leq j < k \leq n$.

The string $u$ can be divided into three substrings $a_1a_2 \ldots, a_j, a_{j+1} \ldots a_k$ nd $a_{k+1} \ldots\ldots a_m$. Let $x, y, z$ represent these strings respectively. Now, as $\leq n$, $|xy| \leq n$ and $u = xyz$. The path with ath value $u$ in the ansition diagram of $M$ is shown in fig. 2.5.

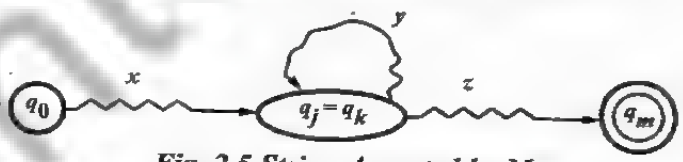



***Fig. 2.5 String Accepted by M***

The automaton $M$ starts from the initial state, i.e., $q_0$. On applying the tring $x$, it reaches $q_j$ (which is equal to $q_k$). On applying the string $y$, it comes ack to $q_j$. So after application of $y^i$ for each $i \geq 0$, the automaton is in the me state, i.e., $q_j$. On applying $z$, it reaches $q_m$, i.e., a final state. Hence $xy^iz$ $L$. Since every state in $Q_1$ is obtained by applying an input symbol, $y \neq \Lambda$.

Note that the decomposition is valid only for strings of length greater than or ual to the number of states. For such a string $u = xyz$, we can iterate (or repeat) e substring $y$ in $xyz$ as many times as we want and get strings of the form $xy^iz$, hich are longer than $xyz$ and are in $L$. By considering the path from $q_0$ to $q_k$ and en from $q_k$ to $q_m$ (without going through the loop), we obtain a path ending in a nal state with path value $xz$. This corresponds to the case when $i = 0$.

***Q.18. What do you mean by closure properties of regular languages ? tate these properties. State pumping lemma and show that L = {a^i b^i \| i>=1} not a regular language.*** ***(R.G.P.V., Dec. 2015)***

***Ans.*** Refer to Q.12, Q.17 and Prob.26.

***Q.19. Define the application of pumping lemma. (R.G.P.V., Dec. 2002)***

***Ans.*** Pumping lemma can be used to prove that certain sets are not regular. llowing are the steps needed to prove it –

***(i) Step 1*** – Suppose $L$ is regular. Let $n$ be the number of states in e corresponding finite automaton.

***(ii) Step 2*** – Select a string $u$ such that $|u| \geq n$. Using pumping write $u = xyz$, with $|xy| \leq n$ and $|y| > 0$.

*(iii) Step 3* – Find an appropriate integer $i$ such that $xy^iz \notin L$, contradicts our assumption. Hence $L$ is not regular.

Note that, the crucial part of it is to find $i$ such that $xy^iz \notin L$. In some cases, we prove $xy^iz \notin L$ by considering $|xy^iz|$. While in some cases we have to use the structure of strings in $L$.

## NUMERICAL PROBLEMS

***Prob.1. Construct the regular expression over {a, b} such that***

***(i) All words begins with a (ii) All words ends with a***

***(iii) For exactly one 'a' (iv) For atleast 2 a's***

***(v) Should contain atleast one double letter.***

***Sol. (i) All Words Begins with a*** – $a(a + b)^*$

***(ii) All Words Ends with a*** – $(a + b)^*a$

***(iii) For Exactly One 'a'*** – $b^*ab^*$

***(iv) For Atleast 2 a's*** – $b^*ab^*ab^*$

***(v) Should Contain Atleast One Double Letter*** – $(a + b)^*(aa)(a + b)^*$.

***Prob.2. Find regular grammars for the following languages on {a, b}***

***(i) $L = \{w | n_a(w)$ and $n_b(w)$ are both even$\}$***

***(ii) $L = \{w | \{n_a(w) - n_b(w)\} \bmod 3 = 1\}$***

***(R.G.P.V., Dec.***

***Sol. (i) $L = \{w | n_a(w)$ and $n_b(w)$ are both even$\}$***

Required grammar is

$G = (V, T, P, A)$

where, $V = \{A, B, C, D\}$

$T = \{a, b\}$

and $P$ is

$A \rightarrow aB/bD/\in$

$B \rightarrow aA/bC$

$C \rightarrow aD/bB$

$D \rightarrow bA/aC$

***(ii) $L = \{w1(n_a(w) - n_b(w)) \bmod 3 = 1\}$***

Required grammar is

$G = (V, T, P, A)$

where, $V = \{A, B, C, D, E\}$

$T = \{a, b\}$

and P is defined as

$A \rightarrow aB/bC/a$

$B \rightarrow aC/bA$

$C \rightarrow aA/bB/b$



***Prob.3. Write regular expression for the following language –***

***The set of strings over alphabet {a, b, c} containing at least one a and at least one b.*** ***(R.G.P.V., Dec. 2008)***

***Sol.*** The regular expression for the set of all strings over $\{a, b, c\}$ ontaining at least one "$a$" and at least one "$b$" is given by

$$\{(a+b+c)^*a(a+b+c)^*b(a+b+c)^*\} + \{(a+b+c)^*b(a+b+c)^*a(a+b+c)^*\}$$

***Prob.4. Give english description of the language of the following regular xpression –***

$$(0^*\ 1^*)^*\ 000\ (0 + 1)^*$$

***(R.G.P.V., Dec. 2008)***

***Sol.*** The given regular expression is

$$(0^*\ 1^*)^*\ 000\ (0 + 1)^*$$

By using the identity $(P + Q)^* = (P^*Q^*)^* = (P^* + Q^*)^*$ for regular xpression, the above regular expression can be written as follows –

$$(0 + 1)^*\ 000\ (0 + 1)^*$$

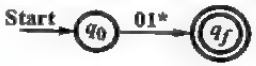
In english, the expression $(0 + 1)^*\ 000\ (0 + 1)^*$ denotes all strings of 0's nd 1's with at least three consecutive 0's.

***Prob.5. Convert the following regular expression to NFA* with transition –***

$$01^*$$ ***(R.G.P.V., Dec. 2008)***

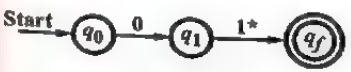
***Sol.*** The given regular expression is $01^*$.

We can express it with the initial state $q_0$ and final $q_f$.

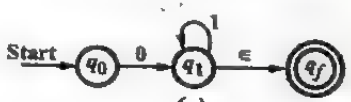



*(a)*

We eliminate the concatenation in the regular expression with the help of tate $q_1$.




*(b)*




*(c)*

*Fig. 2.6*

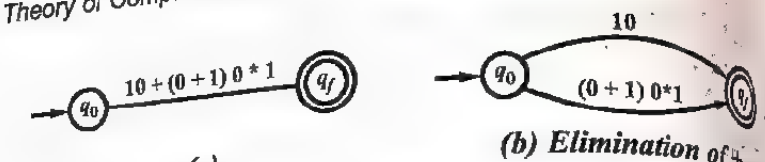
This is the required NFA with $\in$ move.

***Prob.6. Construct an NFA for the following regular expression –***

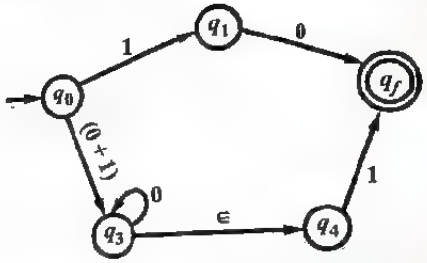
$$10 + (0 + 1)\ 0^*\ 1$$

***(R.G.P.V., Dec. 2007)***

***Sol.*** The NFA is constructed by eliminating the operation +, concatenation nd * in successive steps. The step-by-step construction is given in fig. 2.7.




*(a)* *(b) Elimination of +*




*(c) Elimination of Concatenation and **

*Fig. 2.7*

Now, we eliminate ∈-move in fig. 2.7 (c) and get fig. 2.8 which gi NFA equivalent to the given regular expression.

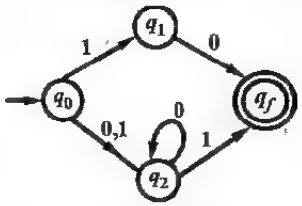


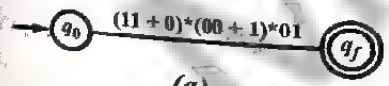

*Fig. 2.8 NFA Equivalent to Given Expression*

***Prob.7. Construct finite automata equivalent to the following regu expression***

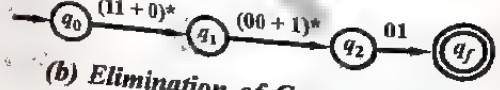
$$(11 + 0)^*(00 + 1)^*01.$$

***(R.G.P.V., Dec.***

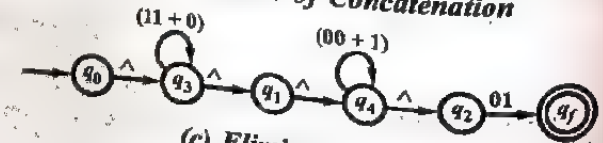
***Sol.*** The finite automata is constructed by eliminating the concate *, and + operations in successive steps. The step-by-step construct given in fig. 2.9.




*(a)*

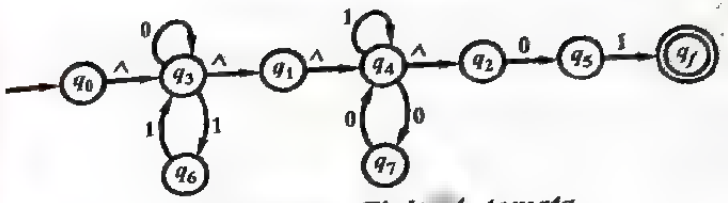



*(b) Elimination of Concatenation*




*(c) Elimination of **





*(d) Equivalent Finite Automata*

*Fig. 2.9*

***Prob.8. Regular expression (aa*)/(bb*) is given. Construct NFA for the cpression and convert this NFA to DFA.*** ***(R.G.P.V., Dec. 2010)***

***Sol.*** The NFA for the given regular expression is given below –

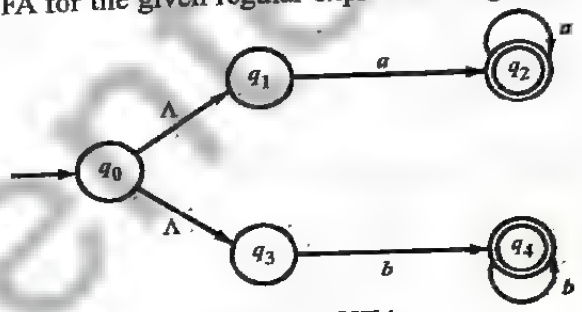



*Fig. 2.10 NFA*

Now, eliminating ∈ in fig. 2.10, we get

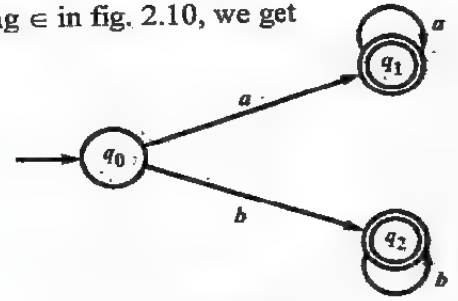



*Fig. 2.11 NFA without ∈*

Now to convert it into DFA, make all possible input transitions to every state. The required DFA is shown in fig. 2.12.

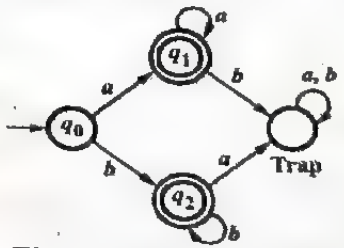



*Fig. 2.12 DFA of aa*/bb**

***Prob.9. Construct DFA for the following set –***

***(i) All strings of {0, 1} such that third symbol from right side is "1".***

***(ii) All strings of {a, b} such that 'aa' is not the sub-string***

***(iii) All strings of {a, b} where number of a's are multiple of and number of b's are multiple of two.***

***(R.G.P.V., June***

***Sol.*** (i) The required regular expression is –

$$(0 + 1)^* 1 (0 + 1) (0 + 1).$$

Now, we construct the transition system and get the NFA accepti set of string. The transition system is shown in fig. 2.13.

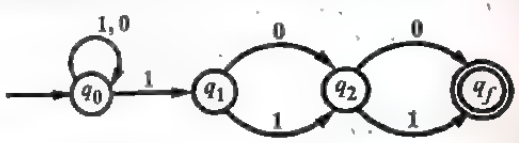



***Fig. 2.13 NFA for (0 + 1)* 1 (0 + 1) (0 + 1)***

The equivalent DFA is given in table 2.1.

**Table 2.1 DFA Equivalent to (0 + 1)* 1 (0 + 1) (0 + 1)**

| *State* | *0* | *1* |
|---|---|---|
| $[q_0]$ | $[q_0]$ | $[q_0q_1]$ |
| $[q_0q_1]$ | $[q_0q_2]$ | $[q_0q_1q_2]$ |
| $[q_0q_2]$ | $[q_0q_f]$ | $[q_0q_1q_f]$ |
| $[q_0q_f]$ | $[q_0]$ | $[q_0q_1]$ |
| $[q_0q_1q_2]$ | $[q_0q_2q_f]$ | $[q_0q_1q_2q_f]$ |
| $[q_0q_1q_f]$ | $[q_0q_2]$ | $[q_0q_1q_2]$ |
| $[q_0q_2q_f]$ | $[q_0q_f]$ | $[q_0q_1q_f]$ |
| $[q_0q_1q_2q_f]$ | $[q_0q_2q_f]$ | $[q_0q_1q_2q_f]$ |

(ii) First create a DFA for the set of all strings of $\{a, b\}$ which contains '$aa$' as substring. The DFA which accepts string that contains '$aa$' as substring is shown in fig. 2.14.

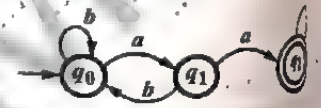



***Fig. 2.14 DFA having "a Substring Over {a, b}***

Now to construct the DFA that does not accept string having '$aa$' as substring, make all non-final states to final states and final states to non-final states.

The resultant DFA which accepts all strings of $\{a, b\}$ such that '$aa$' is not the substring is shown in fig. 2.15.

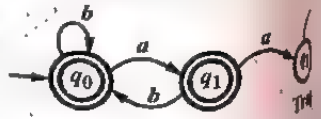



***Fig. 2.15 DFA Accepts St which does not contain " Substring***



(iii) The required DFA is shown in fig. 2.16.

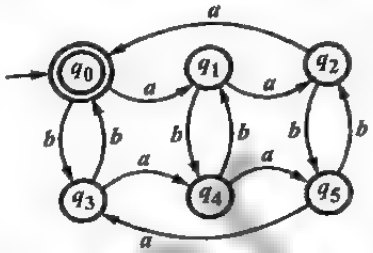


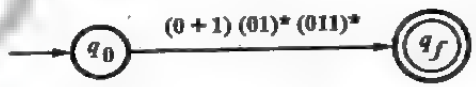

***Fig. 2.16***

***Prob.10. For each of these regular expressions over {0, 1}, draw an NFA with ∈-move recognizing the corresponding language –***

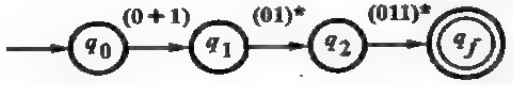
***(i) (0 + 1) (01)* (011)*  (ii) 010* + 0 (01 + 10)*11 .***

***(R.G.P.V., Dec. 2005)***

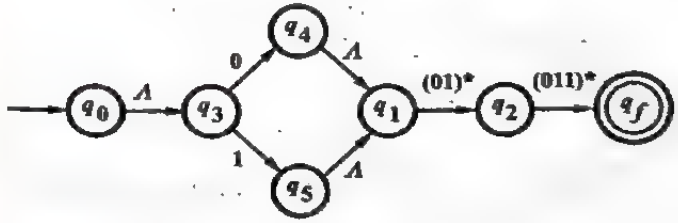
***Sol. (i) (0 + 1) (01)* (011)**** – The NFA with ∈-moves is constructed by liminating the operation +, concatenation and * in successive steps. The tep-by-step construction is given in fig. 2.17.



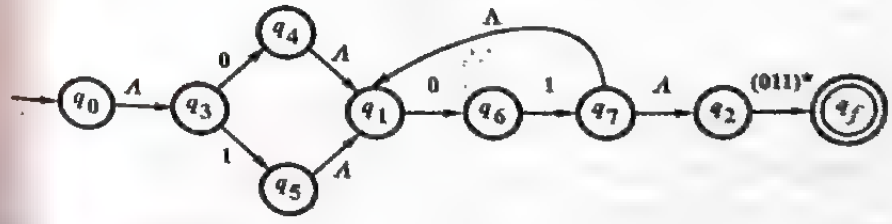

***(a)***



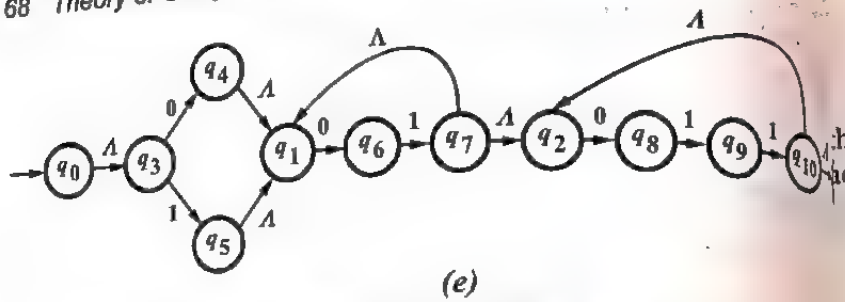

***(b)***




***(c)***



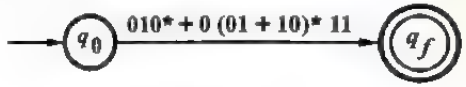

***(d)***



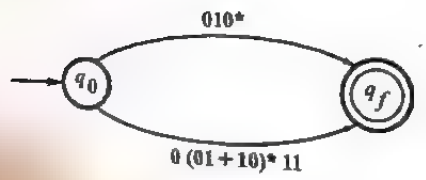
(e)

*Fig. 2.17 Construction of NFA with ∈-moves*

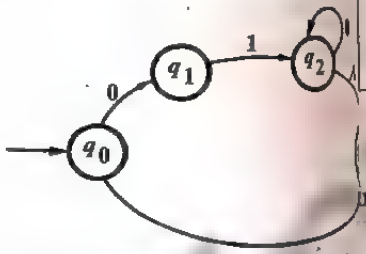
*(ii) 010* + 0 (01 + 10)* 11* – The NFA with ∈-moves is const by eliminating the operation +, concatenation and * in successive step step-by-step construction is given in fig. 2.18.



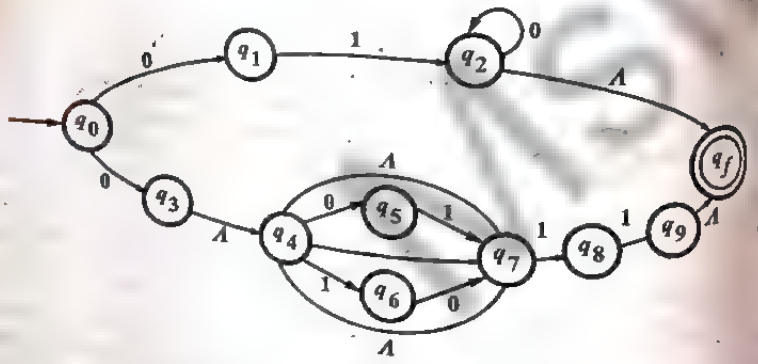
(a)



(b)



(c)



(d)

*Fig. 2.18 Construction of NFA with ∈-moves*

***Prob.11. Construct DFA for the following regular expression –***

***(i) 0 1 (0 + 1)* 10 (ii) ab* (a + b) a.***

***(R.G.P.V., June***



***Sol.*** (i) The given regular expression is 01 (0 + 1)*10. we can construct he transition system with ∈-moves. Eliminating ∈-moves we get the NFA accepting the given set of strings. The transition system is given in fig. 2.19.

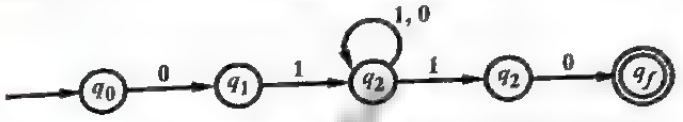


*Fig. 2.19 Transition System for 01 (0 + 1)* 10*

The successor table is constructed for DFA is given in table 2.2.

**Table 2.2**

| *State* | *0* | *1* |
|---|---|---|
| $[q_0]$ | $[q_1]$ | $\phi$ |
| $[q_1]$ | $\phi$ | $[q_2]$ |
| $[q_2]$ | $[q_2]$ | $[q_2q_3]$ |
| $[q_2q_3]$ | $[q_2q_f]$ | $[q_2q_3]$ |
| $[q_2q_f]$ | $[q_2]$ | $[q_2q_3]$ |

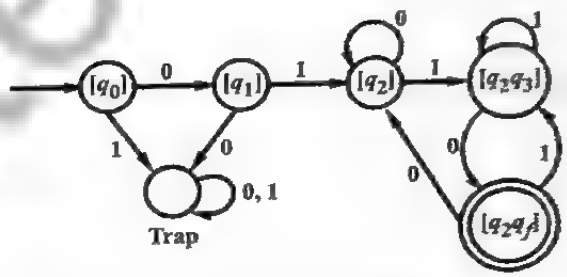


*Fig. 2.20 DFA of 01 (0 + 1)*10*

The state diagram for the successor table is the required DFA as indicated n fig. 2.20. As $q_f$ is the only final state of NFA, $[q_2q_f]$ is only final state of DFA.

***(ii) ab* (a + b) a –***

(a) Construction of transition graph eliminating ∈-moves is shown in fig. 2.21.

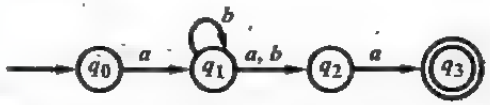


*Fig. 2.21 FA Equivalent to ab* (a + b) a*

(b) Construction of DFA –

**Table 2.3 Transition Table for DFA**

| *State* | *a* | *b* |
|---|---|---|
| $[q_0]$ | $[q_1]$ | $\phi$ |
| $[q_1]$ | $[q_2]$ | $[q_1, q_2]$ |
| $[q_2]$ | $[q_3]$ | $\phi$ |
| $[q_3]$ | $\phi$ | $\phi$ |
| $[q_1, q_2]$ | $[q_2, q_3]$ | $[q_1, q_2]$ |
| $[q_2, q_3]$ | $[q_3]$ | $\phi$ |

As $q_3$ is the final state of NFA, so $[q_2, q_3]$ and $[q_3]$ are the final states of DFA.

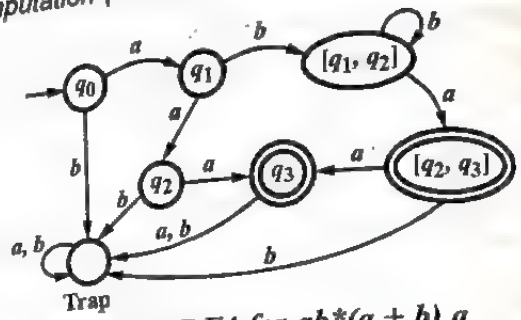



*Fig. 2.22 DFA for ab*(a + b) a*

***Prob.12. Design finite automata for the given regular expression*** *(R.G.P.V., Nov.*
***(a + b)*abb***

***Sol.*** The given regular expression is $(a + b)^*abb$. Now we can construct the transition system with $\in$ moves. Now eliminating these $\in$-moves we get the NFA accepting the given set of strings. The transition system is given in fig. 2.23.

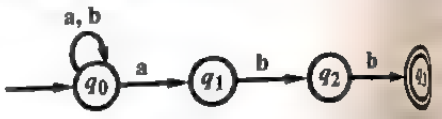



*Fig. 2.23 Transition System for (a + b)**

The successor table for DFA is given in table 2.4.

**Table 2.4**

| *State* | *a* | *b* |
|---|---|---|
| $[q_0]$ | $[q_0, q_1]$ | $[q_0]$ |
| $[q_0, q_1]$ | $[q_0, q_1]$ | $[q_0, q_2]$ |
| $[q_0, q_2]$ | $[q_0, q_1]$ | $[q_0, q_3]$ |
| $[q_0, q_3]$ | $[q_0, q_1]$ | $[q_0]$ |

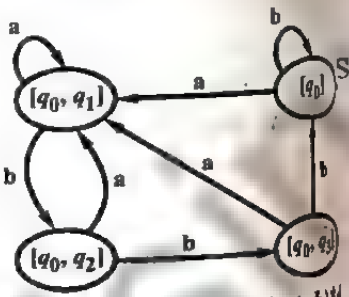



*Fig. 2.24 DFA of (a + b)*a*

The state diagram for the successor table is the required DFA as indicated in fig. 2.24. As $q_3$ is the only final state of NFA, so $[q_0, q_3]$ is only final state of DFA.

***Prob.13. Prove (1 + 00*1) + (1 + 00*1)(0 + 10*1)*(0 + 10*1) = 1(0 + 10*1)**** *(R.G.P.V., Dec.*

***Sol.*** Taking

$$\begin{aligned}
\text{L.H.S.} &= (1 + 00^*1) + (1 + 00^*1)(0 + 10^*1)^*(0 + 10^*1) \\
&= (1 + 00^*1)[\wedge + (0 + 10^*1)^*(0 + 10^*1)] \\
&= (1 + 00^*1)[(0 + 10^*1)^*] \\
&= 1\ (\wedge + 00^*)(0 + 10^*1)^* \\
&= 10^*(0 + 10^*1)^* \\
&= 0^*1\ (0 + 10^*1)^* \\
&= \text{R.H.S.}
\end{aligned}$$



***Prob.14. Construct the NFA for the regular expression (0*(10 + 01)*11)*.***

***Sol.***

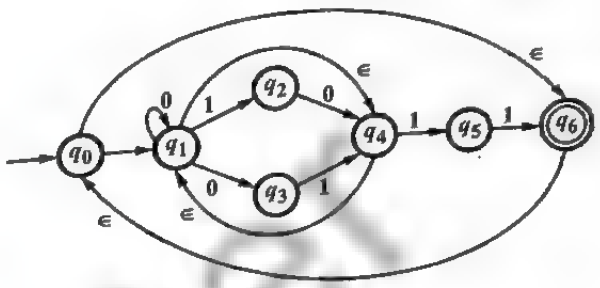



*Fig. 2.25*

***Prob.15. Convert the following automata to regular expression –***

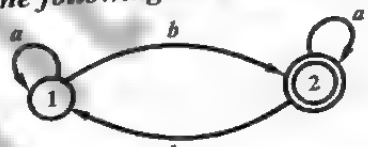



*Fig. 2.26* *(R.G.P.V., Dec. 2007)*

***Sol.*** We can directly apply the algebraic method using Arden's theorem. Since the graph does not contain any $\in$-moves and there is only one initial state. The two equations for $q_1$ and $q_2$ can be written as –

$$q_1 = q_1a + q_2b$$
$$q_2 = q_2a + q_1b$$

By applying Arden's theorem, we get,

$$q_1 = a^*(q_2b) = (q_2b)a^*$$

Substituting $q_1$ in $q_2$, we get,

$$\begin{aligned}
q_2 &= q_2a + (q_2ba^*)b = q_2a + q_2ba^*b \\
&= q_2(a + ba^*b) = (a + ba^*b)^*
\end{aligned}$$

As $q_2$ is the only final state, the regular expression corresponding to the given diagram is $(a + ba^*b)^*$.

***Prob.16. What is the language accepted by the following generalized transition graph ?***

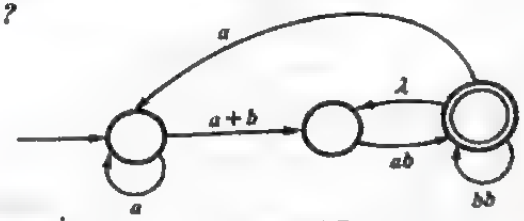



*Fig. 2.27* *(R.G.P.V., Dec. 2004)*

*Sol.* The language accepted by it is

$$L\left(\left(a^*(a+b)\,(\lambda+ab)\,(bb)^*+a\right)^*\right),$$

as should be clear from an inspection of the graph.

***Prob.17. Find out the regular expression for given machine and the theorem used.***

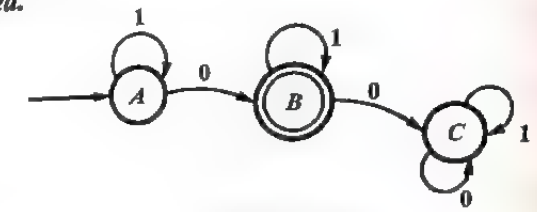



*Fig. 2.28*

***(R.G.P.V., June***

*Sol.* There is only one initial state. Also, there are no $\Lambda$-moves equations are

$$A = A1 + \Lambda$$
$$B = A0 + B1$$
$$C = B0 + C(0 + 1)$$

Since $A = A1 + \Lambda$

By Arden's theorem, we get

$$A = \Lambda 1^* = 1^*$$

So,

$$B = A0 + B1$$
$$= 1^*0 + B1$$

Therefore, by Arden's theorem, we get

$$B = (1^*0)\,1^*$$

Since $B$ is the only final state, we need not solve for $C$, The re expression corresponding to the given diagram is $(1^*0)1^*$.

***Prob.18. Find out the regular expression from given DFA.***

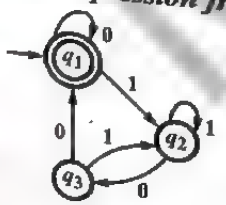



*Fig. 2.29*

***(R.G.P.V., Dec.***

*Sol.* There is no $\Lambda$-moves and only one initial state. We get the follo equations for $q_1$, $q_2$ and $q_3$ –



$$q_1 = q_1 0 + q_3 0 + \Lambda$$
$$q_2 = q_2 1 + q_1 1 + q_3 1$$
$$q_3 = q_2 0$$

By substituting $q_3$ in $q_2$, we get

$$q_2 = q_2 1 + q_1 1 + q_2 01$$
$$q_2 = q_1 1 + q_2(1 + 01)$$

Applying Arden's theorem on $q_2$, we get

$$q_2 = q_1 1(1 + 01)^*$$

By substituting $q_3$ in $q_1$, we get

$$q_1 = q_1 0 + q_2 00 + \Lambda$$

Now, substituting $q_2$ in $q_1$, we get

$$q_1 = q_1 0 + q_1 1\,(1 + 01)^*\,00 + \Lambda$$
$$q_1 = q_1\,(0 + 1\,(1 + 01)^*\,00) + \Lambda$$

Once again by applying Arden's theorem, we get

$$q_1 = \Lambda(0 + 1\,(1 + 01)^*\,00)^*$$
$$q_1 = (0 + 1\,(1 + 01)^*\,00)^*$$

Since $q_1$ is the only final state, so the regular expression corresponding to fig. 2.29 is $(0 + 1\,(1 + 01)^*\,00)^*$

***Prob.19. Convert the following DFA into regular expression***

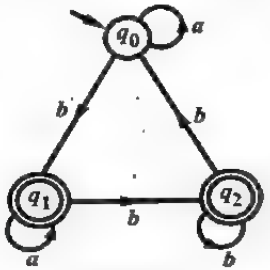



*Fig. 2.30*

***(R.G.P.V., June 2007)***

*Sol.* There is no $\Lambda$-moves and only one initial state. We get the following equations for $q_0$, $q_1$ and $q_2$ –

$$q_0 = q_0 a + q_2 b + \Lambda$$
$$q_1 = q_0 b + q_1 a$$
$$q_2 = q_1 b + q_2 b$$

Applying Arden's theorem on $q_1$ and $q_2$, we get

$$q_1 = q_0 b(a)^*$$
$$q_2 = q_1 b(b)^*$$

Substituting resulting $q_1$ in $q_2$, we get

$$q_2 = q_0\, ba^*bb^*$$

Substituting resulting $q_2$ in $q_0$,

$$q_0 = q_0a + q_0ba^*bb^*b + \Lambda$$
$$= q_0(a + ba^*bb^*b) + \Lambda$$
$$= \Lambda(a + ba^*bb^*b)^*$$
$$= (a + ba^*\ bb^*\ b)^*$$

By substituting $q_0$ in $q_1$, we get

$$q_1 = (a + ba^*bb^*b)^*ba^*$$

Similarly,

$$q_2 = (a + ba^*bb^*b)^*\ ba^*bb^*$$

As there are two final states $q_1$ and $q_2$, thus the required regular expres is obtained by taking union of $q_1$ and $q_2$, Therefore,

$$q_1 + q_2 = \{(a + ba^*bb^*b)^*ba^*\} + \{(a + ba^*bb^*b)^*\ ba^*$$
$$= (a + ba^*bb^*b)^*ba^*(\Lambda + bb^*)$$
$$= \mathbf{(a + ba^*bb^*b)^*ba^*b^*}$$

***Prob.20. Find a regular expression corresponding to a finite auton***

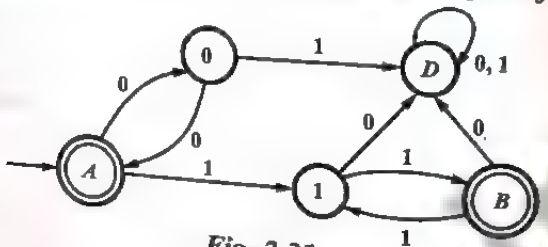



*Fig. 2.31*

***(R.G.P.V., Dec. 2***

***Sol.*** The state labelled $A$ is both the initial state and an accepting st this tells us that $\Lambda \in L$. It is easy to see from the diagram that the other str $x$ for which $\delta^*(A, x) = A$ are even-length strings of 0's, corresponding to regular expression $(00)^*$.

No matter what input we get from here on, the result will never be in We denote this case by state $D$. Once the FA reaches $D$, it stays in that st a string $x$ for which $\delta^*(A, x) = D$ cannot be a prefix of any element of $L$

The easiest way to reach the other accepting state $B$ from $A$ is with string 11. Once the FA is state $B$, any even-length string of 1's returns it to and these are the only strings that do this. Therefore, if $x$ is a string that caus the FA to go from $A$ to $B$ without revisiting $A$, $x$ must be of the form $11(1$ for some $k \geq 0$. Since such a string could be preceded by $(00)^j$, the langua

$$\{x \mid \delta^*(A, x) = B\}$$

corresponds to the regular expression $(00)^*\ 11(11)^*$. The language $L$, therefo



corresponds to the regular expression $(00)^* + (00)^*11(11)^*$, which can be simplified to

$$(00)^*\ (11)^*\ .$$

***Prob.21. Convert the following DFA into regular expression.***

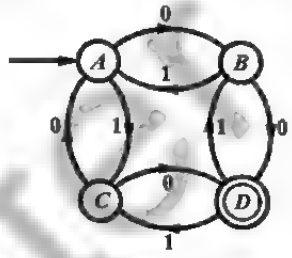



*Fig. 2.32*

***(R.G.P.V., June 2007)***

***Sol.*** As the graph does not contain $\Lambda$-moves and there is only one initial state, we get the following equations for $A$, $B$, $C$ and $D$ –

$$A = B1 + C0 + \Lambda$$
$$B = A0 + D1$$
$$C = A1 + D1$$
$$D = B0 + C0$$

By substituting $B$ and $C$ in $A$ equation,

$$A = A01 + D11 + A10 + D10 + \Lambda$$
$$= (D11 + D10 + \Lambda) + A\ (01 + 10)$$
$$= (D\ (11 + 10) + \Lambda)\ (01 + 10)^*$$
$$= D\ (11 + 10)\ (01 + 10)^* + (01 + 10)^*$$

By substituting resulting $A$ in $B$,

$$B = (D\ (11 + 10)\ (01 + 10)^* + (01 + 10)^*)\ 0 + D1$$
$$= D\ (11 + 10)\ (01 + 10)^*\ 0 + (01 + 10)^*\ 0 + D1$$
$$= D\ ((11 + 10)\ (01 + 10)^*\ 0 + 1) + (01 + 10)^*0$$

By substituting resulting $A$ in $C$,

$$C = (D\ (11 + 10)\ (01 + 10)^* + (01 + 10)^*)1 + D1$$
$$= D\ (11 + 10)\ (01 + 10)^*1 + (01 + 10)^*1 + D1$$
$$= D\ ((11 + 10)\ (01 + 10)^*1 + 1) + (01 + 10)^*1$$

Putting resulting $B$ and $C$ in $D$, we have

$$D = \{D\ ((11 + 10)\ (01 + 10)^*\ 0 + 1) + (01 + 10)^*0\}\ 0$$
$$+ \{D\ ((11 + 10)\ (01 + 10)^*\ 1 + 1) + (01 + 10)^*\ 1\}0$$
$$= D\ ((11 + 10)\ (01 + 10)^*\ 0 + 1)0 + (01 + 10)^*00$$
$$+ D\ ((11 + 10)\ (01 + 10)^*\ 1 + 1)\ 0 + (01 + 10)^*\ 10$$
$$= D\ \{((11 + 10)\ (01 + 10)^*\ 0 + 1)\ 0 + (01 + 10)^*$$
$$00 + ((11 + 10)\ (01 + 10)^*\ 1 + 1)0$$
$$+ (01 + 10)^*\ 10\}$$

Finally by applying Arden's theorem, we get

$$D = (((11+10)(01+10)^*0+1)0+(01+1$$
$$+((11+10)(01+10)^*1+1)0+(01+1$$

***Prob.22. Construct minimum state DFA for the ahead machine, minimization algorithm.***

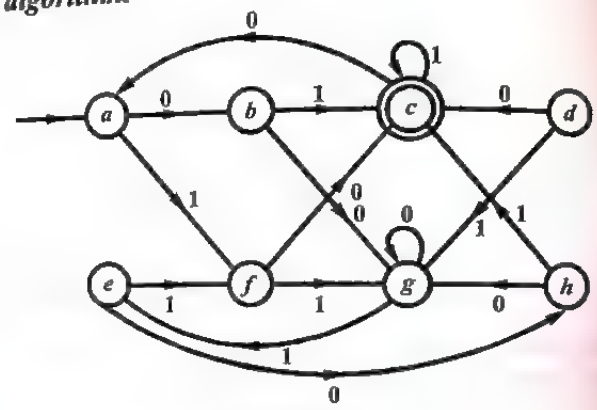



*Fig. 2.33*

***(R.G.P.V., Dec. 2006, June***

***Sol.*** Let $M$ be the finite automaton of fig. 2.33. In fig. 2.34 we constructed a table with an entry for each pair of states. An × is placed table each time we discover a pair of states that cannot be equivalent. Ini an × is placed in each entry corresponding to one final state and one non state. We place an × in the entries $(a, c)$, $(b, c)$, $(c, d)$, $(c, e)$, $(c, f)$, $(c, g)$ $(c, h)$.

Next for each pair of states $p$ and $q$ that are not already known distinguishable we consider the pairs of states $r = \delta(p, a)$ and $s = \delta(q,$ each input symbol $a$. If states $r$ and $s$ have been shown to be distinguish by some string $x$, then $p$ and $q$ are distinguishable by string $ax$. Thus i entry $(r, s)$ in the table has an ×, and × is also placed at the entry $(p, q)$. I entry $(r, s)$ does not yet have an ×, then the pair $(p, q)$ is placed on associated with the $(r, s)$-entry. At some future time, if the $(r, s)$ entry rec an ×, then each pair on the list associated with the $(r, s)$-entry also receives

Continuing, we place an × in the entry $(a, b)$, since the entry $(\delta(b, 1)$ $1)) = (c, f)$ already has an ×. Similarly, the $(a, d)$-entry receives an × since entry $(\delta(a, 0), \delta(d, 0)) = (b, c)$ has an ×. Consideration of the $(q, e)$-entr input 0 results in the pair $(a, e)$ being placed on the list associated with ( Observe that on input 1, both $a$ and $e$ go to the same state $f$ and hence no starting with a 1 can distinguish $a$ from $e$. Because of the 0-input, the pair is placed on the list associated with $(b, g)$. When the $(b, g)$-entry is consi it receive an × on account of a 1-input, and hence the pair $(a, g)$ receives since it was on the list for $(b, g)$. The string 01 distinguishes $a$ from $g$.



On completion of the table in fig. 2.34, we conclude that the equivalent states are $a \equiv e$, $b \equiv h$, and $d \equiv f$. The minimum-state finite automaton is given fig. 2.35.

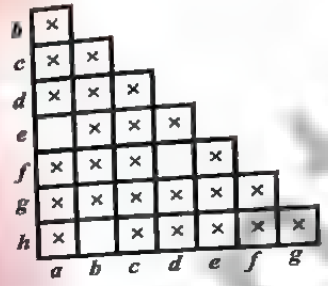



*Fig. 2.34 Calculation of Equivalent States*

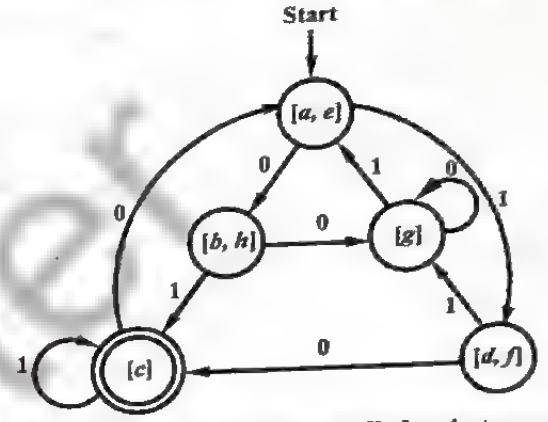



*Fig. 2.35 Minimum State Finite Automaton*

***Prob.23. Find out whether the following grammars generate the same language –***

$G_1$ –
$A \to 0B \mid 1E$
$B \to 0A \mid 1F \mid \in$
$C \to 0C \mid 1A$
$D \to 0A \mid 1D \mid \in$
$E \to 0C \mid 1A$
$F \to 0A \mid 1B \mid \in$

$G_2$ –
$X \to 0Y \mid 0 \mid 1Z$
$Y \to 0X \mid 1Y \mid 1$
$Z \to 0Z \mid 1X$

***(R.G.P.V., June 2009)***

***Sol.*** Since the grammars $G_1$ and $G_2$ are the regular grammars, $L(G_1) = L(G_2)$, if the minimal state automata accepting $L(G_1)$, and the minimal state automata accepting $L(G_2)$ are identical. The transition diagram of the automata accepting $L(G_1)$ is given in fig. 2.36.

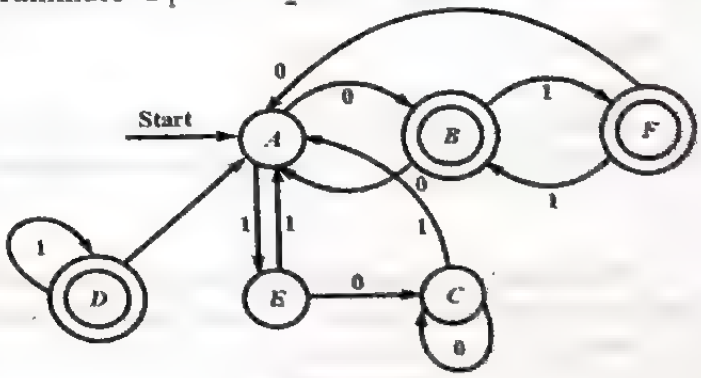



*Fig. 2.36*

The automata is deterministic, hence to minimize it we proceed as fo

Since state D is unreachable state, first eliminate state D, hen eliminating state D we get –

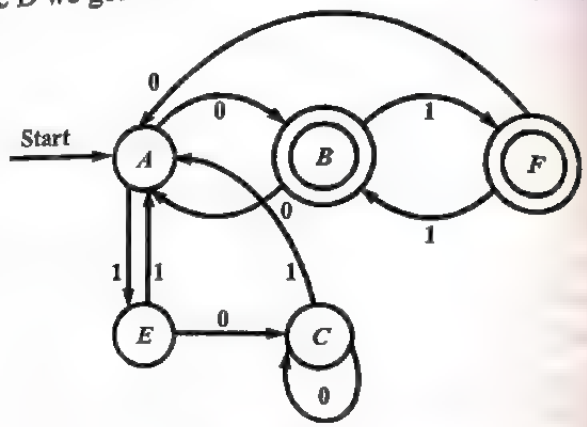



*Fig. 2.37*

We then identify the non-distinguishable states if any in the above aut as follows –

Initially we have two groups as follows –

| A, E, C | B, F |
|---|---|
| Group I | Group II |

Since,
$$\delta(A, 0) = B$$
$$\delta(E, 0) = C$$
$$\delta(C, 0) = C$$

State $B$ is distinguishable from rest of the members of group I. Here is – divide group I into two groups – one containing $A$, and other containing as shown below –

| A | E, C | B, F |
|---|---|---|
| Group I | Group II | Group III |

Since,
$$\delta(E, 0) = C$$
$$\delta(C, 0) = C$$

Partitioning of group II is not possible because the transitions from all members of group II goes to group II only for input 0. Similarly,
$$\delta(E, 1) = A$$
$$\delta(C, 1) = A$$

Partitioning of group II is not possible, because the transitions from the members of group II goes to group I only for input 1.

Since,
$$\delta(B, 0) = A$$
$$\delta(F, 0) = A$$



Partitioning of group III is not possible, because the transitions from all members of group III goes to group I only for input 0. Similarly,
$$\delta(B, 1) = F$$
$$\delta(F, 1) = B$$

Partitioning of group III is not possible, because the transitions from all members of group III goes to group III only for input 1.

Hence states $E$ and $F$ are non-distinguishable states, as well as states and $F$ are also non-distinguishable states. Therefore, if we merge $E$ and $C$ to orm a state $E_1$, $B$ and $F$ to from $B_1$, we get fig. 2.38.

Since there exists no dead state in the automata shown above, it is a inimal state automata accepting $L(G_1)$.

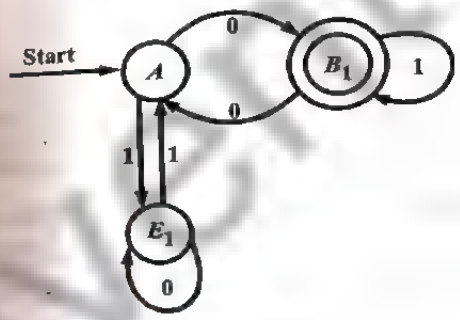



*Fig. 2.38*

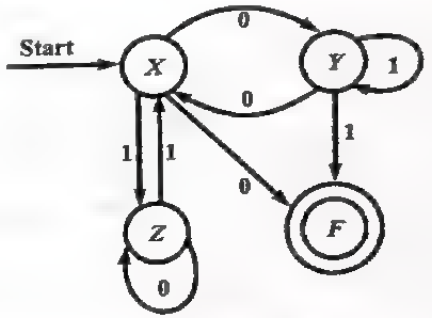



*Fig. 2.39*

The transition diagram of the automata accepting $L(G_2)$ is fig. 2.39. This is a non-deterministic automata, its equivalent deterministic automata is –

| | 0 | 1 |
|---|---|---|
| {X} | {Y, F} | {Z} |
| *{Y, F} | {X} | {Y, F} |
| {Z} | {Z} | {X} |

Therefore, the transition diagram of the deterministic automata is shown in fig. 2.40.

This automata does not contain, unreachable, non-distinguishable, as well as dead state, hence it is a minimal state automata accepting $L(G_2)$, and it is identical to the minimal state automata accepting $L(G_1)$, therefore, $L(G_1) = L(G_2)$ and thus $G_1$ and $G_2$ generate the same language.

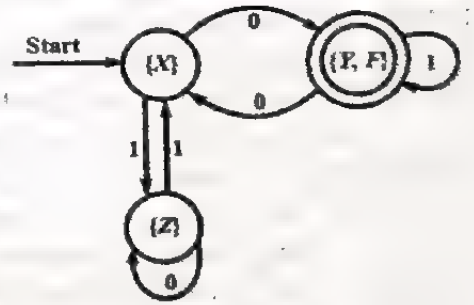



*Fig. 2.40*

***Prob.24. Show that the set*** $L = \left\{b^{i^2} \mid i \geq 1\right\}$ ***is not regular.***

***(R.G.P.V., Jun***

***Sol. (i) Step 1*** – Suppose $L$ is regular and we get a contradiction. Let $n$ be the number of states in FA accepting $L$.

***(ii) Step 2*** – Let $u = b^{n^2}$. Then $|u| = n^2 > n$. By pumping lemma we can write $u = xyz$ with $|xy| \leq n$ and $|y| > 0$.

***(iii) Step 3*** – Consider $xy^2z$. $|xy^2z| = |x| + 2|y| + |z| > |x| + |y| + |z|$ as $|y| > 0$. It means $n^2 = |xyz| = |x| + |y| + |z| < |xy^2z|$. But $|xy| \leq n, |y| \leq n$. Thus, $|xy^2z| = |x| + 2|y| + |z| \leq n^2 + n$, i.e.,

$$n^2 < |xy^2z| \leq n^2 + n < n^2 + n + n + 1$$

Hence, $|xy^2z|$ strictly occurs between $n^2$ and $(n+1)^2$, but is not equal to any one of them. Thus, $|xy^2z|$ is not a perfect square. So $xy^2z \notin L$ by pumping lemma $xy^2z \in L$. This is a contradiction. ***Thus, L is not regular.***

***Prob.25. Prove*** $L = \{a^p \mid p$ ***is a prime\}*** ***is not regular using pumping lemma.***

***(R.G.P.V., Dec.***

***Sol. (i) Step 1*** – We suppose $L$ is regular and get a contradiction. Let $n$ be the number of states in the finite automaton accepting $L$.

***(ii) Step 2*** – Let $p$ be a prime number greater than $n$, Let $u = a^p$. By pumping lemma, $u$ can be written as $u = xyz$, with $|xy| \leq n$ and $|y| > 0$. $x, y, z$ are simply strings of $a$'s. So, $y = a^m$ for some $m \geq 1$ (and $\leq n$).

***(iii) Step 3*** – Let $i = p + 1$. Then $|xy^iz| = |xyz| + |y^{i-1}| = p + (i-1)m = p + pm$. By pumping lemma, $xy^iz \in L$. But $|xy^iz| = p + pm = p(1+m)$ and $p(1+m)$ is not a prime. So $xy^iz \notin L$. This is a contradiction. ***Thus L is not regular.***

***Prob.26. Show that*** $L = \{0^i 1^i / i \geq 1\}$ ***is not regular.***

***Sol. (i) Step 1*** – Suppose $L$ is regular and we get a contradiction. Let $n$ be the number of states in finite automaton accepting $L$.

***(ii) Step 2*** – Let $u = 0^n 1^n$. Then $|u| = 2n > n$. By pumping lemma we write $u = xyz$ with $|xy| \leq n$ and $|y| \neq 0$.

***(iii) Step 3*** – We want to find $i$ so that $xy^iz \notin L$ for getting a contradiction. The string $y$ can be in any of the following forms –

**(a) Case 1** – $y$ has only 0's, i.e., $y = 0^k$ for some $k \geq 1$.

**(b) Case 2** – $y$ has only 1's, i.e., $y = 1^l$ for some $l \geq 1$.

**(c) Case 3** – $y$ has both 0's and 1's, i.e., $y = 0^k1^j$ for some $k, j \geq 1$.



In case 1, we can take $i = 0$. As $xyz = 0^n1^n$, $xz = 0^{n-k}1^n$. As $k \geq 1$, $n - k \neq n$. So $xz \notin L$.

In case 2, we take $i = 0$. As before, $xz$ is $0^n1^{n-l}$ and $n \neq n - l$. So $xz \notin L$.

In case 3, we take $i = 2$. As $xyz = 0^{n-k}0^k1^j1^{n-j}$, $xy^2z = 0^{n-k}0^k1^j0^k1^j1^{n-j}$. As $xy^2z$ is not of the form $0^i1^i$, $xy^2z \notin L$.

Thus in all the cases we get a contradiction. ***Therefore, L is not regular.***

***Prob.27. Prove the following is not regular language –***

$\{0^n1^n \mid n \geq 1\}$

***This language consisting of a string of 0's followed by an equal length string of 1's, is the language*** $L_{01}$ ***(R.G.P.V., Dec. 2008)***

***Sol.*** Refer to Prob.26.

***Prob.28. Use pumping lemma to prove that the following sets are not regular.***

***(i)*** $L = \{a^n b^{2n} \mid n \geq 0\}$ ***(ii)*** $L\ \{a^n \mid n$ ***is a prime number\}.***

***(R.G.P.V., June 2007)***

***Sol. (i)*** $L = \{a^n b^{2n} \mid n \geq 0\}$ – We prove $L$ is not regular by contradiction. If $L$ is regular, we can apply pumping lemma. Let $n$ be the number of states. Let $w = a^nb^{2n}$. By pumping lemma, $w = xyz$ with $|xy| \leq n$, $|y| > 0$ and $xz \in L$. As $|xy| \leq n$, $xy = a^m$ and $y = a^l$ where $0 < l \leq n$. So $xz = a^{n-l}b^{2n} \in L$, a contradiction since $n - l \neq n$. Thus $L$ is not regular.

***(ii)*** $L\ \{a^n \mid n$ ***is a prime number\}*** – Refer to Prob.25.

***Prob.29. Apply pumping lemma to prove that*** $L = \{a^nb^{2n} \mid n \geq 1\}$ ***is not regular set.*** ***(R.G.P.V., Dec. 2003)***

***Sol. (i) Step 1*** – Suppose $L$ is regular and we get a contradiction. Let $m$ be the number of states in finite automaton $M$ accepting $L$.

***(ii) Step 2*** – Let $u = a^mb^{2m}$. Then $|u| = 3m > m$. By pumping lemma we write $u = xyz$ with $|xy| \leq n$ and $|y| \neq 0$.

***(iii) Step 3*** – We want to find $i$ so that $xy^iz \notin L$ for getting a contradiction. The string $y$ can be in any of the following forms –

**Case 1** – $y$ has only $a$'s, i.e., $y = a^k$ for some $k \geq 1$.

**Case 2** – $y$ has only $b$'s, i.e., $y = b^l$ for some $l \geq 1$.

**Case 3** – $y$ has both $a$'s and $b$'s, i.e., $y = a^kb^j$ for some $k, j \geq 1$.

In case 1, we can take $i = 0$. As $xyz = a^mb^{2m}$, $xz = a^{m-k}b^{2m}$. As $k \geq 1$, $m - k \neq m$. So $xz \notin L$.

In case 2, we take $i = 0$. As before $xz$ is $a^mb^{2m-l}$ and $2m - l \neq 2m$ as $l \geq 1$. So $xz \notin L$.

∴ In case 3, we take $i = 2$. As $xyz = a^{m-k}a^kb^jb^{2m-j}$ ∴ $a^{m-k}a^kb^jb^{2m-j}$. As $xy^2z$ is not of the form $a^nb^{2n}$, $xy^2z \notin L$.

Thus, in all cases we get a contradiction. ***Therefore, L is not reg***

***Prob.30. Prove whether the sets are regular or not –***

***(i) $\{0^{2n} \mid n \geq 1\}$ (ii) $\{0^n 1^n \mid n \geq 0\}$.***

***(R.G.P.V., Dec.***

***Sol. (i)*** $\{0^{2n} \mid n \geq 1\}$ – We can write $0^{2n}$ as $0\,(0^2)^i0$, where $i \geq 0$. Now $\{(0^2)^i \mid i \geq 0\}$ is simply $\{0^2\}^*$, so $L$ is represented by the regular expression $0\,(p)^*0$, where $p$ represents $\{0^2\}$. The corresponding finite automata is given in fig. 2.41.

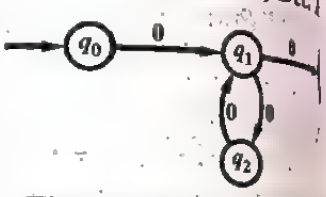



***Fig. 2.41 FA of*** $\{0^{2n} \mid n$

***(ii)*** $\{0^n 1^n \mid n \geq 0\}$ – The proof is by contradiction. Let $L =$ $\mid n \geq 0\}$. $\{\delta(q_0, 0^n) \mid n \geq 0\}$ is a subset of $Q$ and hence finite, so $\delta(q_0,$ $\delta(q_0, 0^m)$ for some $m$ and $n$, $m \neq n$. So $\delta(q_0, 0^m1^n) = \delta(\delta(q_0, 0^m), 1^$ $(\delta(q_0, 0^n), 1^n) = \delta(q_0, 0^n1^n)$. As $0^n1^n \in L$, $\delta(q^0, 0^n1^n)$ is a final state a is $\delta(q_0, 0^m 1^n)$ This means $0^m1^n \in L$ with $m \neq n$, which is a contradi Hence $L$ is not regular.

***Prob.31. Using pumping lemma show that the following set are not reg***

***(i) $\{a^n b^{2n} \mid n > 0\}$ (ii) $\{a^n b^m \mid 0 < n < m\}$.***

***(R.G.P.V., Dec.***

***Sol.*** (i) Let $i$ be constant of pumping lemma.

Select $Z = a^ib^{2i}$, this ensures that $Z$ is in $L$ and $|Z| \geq i$.

If we write $Z = uvw$, then the possible choices of $v$ satisfying the condit

$1 \leq |v| \leq i$ and $|uv| \leq i$ are –

$v = a^p$, where $1 \leq p \leq i$, i.e., $v$ can be chosen to be string of min one $a$ and maximum $i$ $a$'s

For this choice of $v$, $uv^0w$ will be $a^{i-p}b^i$, and since $1 \leq p \leq i$, $uv^0$ have $(i-p)$ $a$'s and $(2i)$ $b$'s, hence it cannot belong to $L$. Therefore, w contradiction to pumping lemma.

$v$ cannot contain $b$'s, because if $v$ is chosen to contain $b$'s, then $|uv|$ not be less than or equal to $i$. Thereby proving that $L$ is not regular.

(ii) Let $n$ be constant of pumping lemma.

Select $Z = a^nb^m$, this ensures that $Z$ is in $L$, and $|Z| \geq n$ (because $n$ conditions –

If we write $Z = uvw$, then the possible choices of $v$ satisfying

$1 \leq |v| \leq n$ and $|uv| \leq n$ are –

$v = a^p$, where $1 \leq p \leq n$, i.e., $v$ can be chosen to be string of min one $a$ and maximum $n$ $a$'s.

For this choice of $v$, $uv^2w$ will be $a^{n+p}b^n$, and since $1 \leq p \leq n$, $uv^2w$ will have more number of $a$'s than $b$'s, hence it cannot belong to $L$. Therefore, we get contradiction to pumping lemma.

$v$ cannot contain $b$'s, because if $v$ is chosen to contain $b$'s, then $|uv|$ will not be less than or equal to $n$. Thereby proving that $L$ is not regular.

***Prob.32. Prove that $a^n \subset b^n | n \geq 1$ is not regular.***

***(R.G.P.V., Dec. 2011)***

***Sol.*** Let $u$ be any string such that

$u \in a^n \subset b^n$ and $|u| \geq 3$

Now, let us assume that

$u = xyz$ where $|y| \neq 0$ and $|xy| \leq n$

and according to pumping lemma

If u is regular the $xy^iz$ is also regular and vice versa.

Here, 6 cases arises –

**Case-1** – Let $y = c$

$x = a$

$z = b$

$u = xy^iz = a(c)^ib$

When $i = 1$, $u = acb$, which is in $L$.

When $i = 2$, $u = accb$, which is not in $L$.

Hence the language is not regular.

**Case-2** – Let $y = a$

$x = \epsilon$

$z = cb$

$u = xy^iz = (a)^icb$

When $i = 1$, $u = acb$, which is in $L$.

When $i = 2$, $u = aacb$, which is not in $L$.

Hence the language is not regular.

**Case-3** – Let $y = b$

$x = ac$

$z = \epsilon$

$u = ac(b)^i$

When $i = 1$, $u = acb$, which is in $L$.

When $i = 2$, $u = acbb$, which is not in $L$.

**Case-4** – Let $y = ac$

$x = \epsilon$

$z = b$

$u = (ac)^ib$

When $i = 1$, $u = acb$, which is in $L$.

When $i = 2$, $u = acacb$, which is not in $L$.

Hence, given language is not regular.

Similarly for case-5 as $y = cb$, $z = \in$, $x = a$ the given language regular for $i = 2$.

**Case-6** – Let $y = acb$
$x = \in \& z = \in$
$u = (acb)^i$

When $i = 1$, $u = acb$, which is in $L$.
When $i = 2$, $u = acbacb$, which is not in $L$.
Hence, the given language is not regular.

It is clear from all the above cases that the language $L = \{a^n \subset b^n\}$ is not regular.

***Prob.33. Explain pumping lemma for regular sets. Also prove following set is not regular.***

$$L = \{0^m 1^n 0^{m+n} |\ m \geq 1 \text{ and } n \geq 1\}$$ ***(R.G.P.V., Dec.)***

***Sol.*** **Pumping Lemma** – Refer to Q.17.

Given set, $L$ is $\{0^m 1^n 0^{m+n} | m \geq 1 \text{ and } n \geq 1\}$

$L = \{0100, 011000, 001000, 00110000,.....\}$

**Case I** – Let, $y = 1$ $(\because y \neq \wedge)$
$x = 0$
$z = 00$
$u = 0(1)^i 00$

When $i = 1$, $u = 0100$ which is in '$L$'
When $i = 2$, $u = 01100$ which is not in '$L$'
Therefore, $L = \{0^m 1^n 0^{m+n} |\ m \geq 1, n \geq 1\}$ is not regular.

**Case 2** – Let, $y = 0$
$x = 01$
$z = 0$
$u = 01(0)^i 0$

When $i = 1$, $u = 0100$ which is in '$L$'
When $i = 2$, $u = 01000$ which is not in '$L$'
Therefore, $L = \{0^m 1^n 0^{m+n} | m \geq 1, n \geq 1\}$ is not regular.

**Case 3** – Let, $y = 10$
$x = 0$
$z = 0$
$u = 0(10)^i 0$

When $i = 1$, $u = 0100$ which is in '$L$'
When $i = 2$, $u = 010100$ which is not in '$L$'
Therefore, $L = \{0^m 1^n 0^{m+n} |\ m \geq 1, n \geq 1\}$ is not regular.



## APPLICATIONS OF FINITE AUTOMATA, MINIMIZATION OF FSA

***Q.20. What is DFA and its applications ?***

***Or***

***Define DFA. List three household applications of finite automata.*** ***(R.G.P.V., Dec. 2015)***

***Ans.*** **DFA** – Refer to Q.7 (Unit-I).

There are a variety of software design problems that are simplified by utomatic conversion of regular expression notation to an efficient computer mplementation of the corresponding finite automaton. Two such applications re as follows –

***(i) Lexical Analyzers*** – The tokens of a programming language are lmost without exception expressible as regular sets.

***(ii) Text Editors*** – Certain text editors and similar programs permit he substitution of a string for any string matching a given regular expression.

***(iii) Process Scheduler*** – FA allocates resource to the process ccording to the scheduling criteria process.

***Q.21. What is the use of finite automata in Lexical analyzer ?***

***Ans.*** In Lexical analyzers, tokens are almost without exception expressible s regular sets. For example,

In ALGOL

identifiers = (letter) (letter + digit)*

Since identifiers are upper or lower case letter followed by any sequence f letter & digit so that we can express it as regular set. A number of Lexical nalyzer generators take as input a sequence of regular expressions describing he tokens and produce a single finite automata that recognises any token. Generally they convert regular expression to NFA with $\in$-transition and then roduce DFA directly without eliminating $\in$-transition before. In DFA, every inal state indicates that a particular token is found. This shows that the utomation is done on Moore machine which gives some output when any oken found. This Lexical analyzer is then used in compiler.

***Q.22. Give method of constructing minimum automaton.***

***Ans.*** A step-by-step method for the construction of minimum automaton s as follows –

***(i) Step 1 (Construction of $\pi_0$)*** – By definition of 0-equivalence, $_0 = \{Q_1^0, Q_2^0\}$, where $Q_1^0$ is the set of all final states and $Q_2^0 = Q - Q_1^0$ e., the set of remaining all non-final states.

***(ii) Step 2 (Construction of $\pi_{k+1}$ from $\pi_k$)*** – Let $Q_i^k$ be ... in $\pi_k$. If $q_1$ and $q_2$ are in $Q_i^k$, they are $(k+1)$-equivalent (if $\delta(q_1, a)$ ... , $a$) are $k$-equivalent). Find out, whether $\delta(q_1, a)$ and $\delta(q_2, a)$ are in ... equivalence class in $\pi_k$ for every $a \in \Sigma$. If yes, $q_1$ and ... $(k+1)$-equivalent. Thus, in this way $Q_i^k$ is further divided into $(k+1)$-equ... classes. Repeat this step for every $Q_i^k$ in $\pi_k$ to get all the elements of ...

***(iii) Step 3 (Construction of $\pi_n$)*** – Construct $\pi_n$ for $n = 1$ ..... until $\pi_n = \pi_{n+1}$.

***(iv) Step 4 (Construction of Minimum Automaton)*** – ... required minimum state automaton, the states are the equivalence ... (obtained in step 3), i.e., the elements of $\pi_n$. The state table can be obta... replacing a state $q$ by the corresponding equivalence class $[q]$.

The construction of $\pi_0$, $\pi_1$, etc., is easy when the transition ... given. $\pi_0 = \{Q_1^0, Q_2^0\}$, where $Q_1^0 = F$ and $Q_2^0 = Q - F$. The subset in ... be obtained by partitioning subsets of $\pi_0$ further. If $q_1$, $q_2 \in Q_i$, ... consider the states in each $a$-column, where $a \in \Sigma$ corresponding to ... $q_2$. If $q_1$ and $q_2$ are in the same subset of $\pi_0$, then they are 1-equivalent ... states under some $a$-column are in different subsets of $\pi_0$, then $q_1$ and ... not 1-equivalent. Generally, $(k+1)$-equivalent states are obtained by ... the above method for $q_1$ and $q_2$ in $Q_i^k$.

## NUMERICAL PROBLEMS

***Prob.34. Consider the FA below and construct the, smallest DFA ... accepts the same language.***

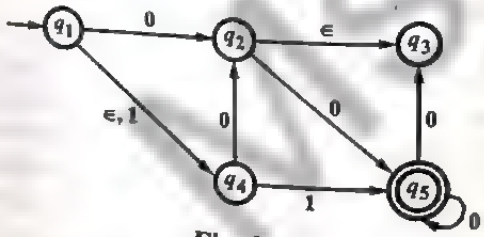



*Fig. 2.42* *(R.G.P.V., Dec...*

***Sol.*** The given FA is redrawn as –

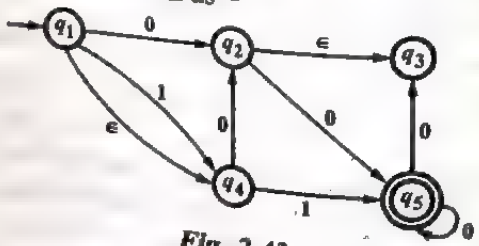



*Fig. 2.43*



In this transition diagram, there are ∈ moves, so we have to remove ∈ moves. We first remove ∈ move from $q_1$ to $q_4$ to get fig. 2.44.

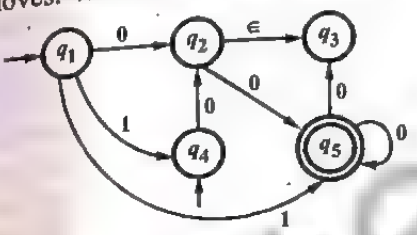



*Fig. 2.44*

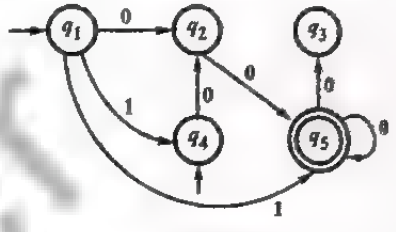



*Fig. 2.45*

Now, we eliminate the ∈ move from $q_2$ to $q_3$ in fig. 2.44 to get fig. 2.45.

Now, we construct the transition table as given in table 2.5 corresponding fig. 2.45.

**Table 2.5**

| *States/Σ* | *0* | *1* |
|---|---|---|
| → $q_1$ | $q_2$ | $q_4, q_5$ |
| $q_2$ | $q_5$ | – |
| $q_3$ | – | – |
| → $q_4$ | $q_2$ | – |
| (*$q_5$*) | $q_3, q_5$ | – |

The successor table is constructed for DFA and given in table 2.6.

**Table 2.6**

| *Q* | *$Q_0$* | *$Q_1$* |
|---|---|---|
| → $[q_1]$ | $[q_2]$ | $[q_4, q_5]$ |
| $[q_2]$ | $[q_5]$ | $\phi$ |
| → $[q_4, q_5]$ | $[q_2, q_3, q_5]$ | $\phi$ |
| $[q_5]$ | $[q_3, q_5]$ | $\phi$ |
| $[q_2, q_3, q_5]$ | $[q_3, q_5]$ | $\phi$ |
| $[q_3, q_5]$ | $[q_3, q_5]$ | $\phi$ |

The state diagram for the successor table is the required DFA as shown in fig. 2.46.

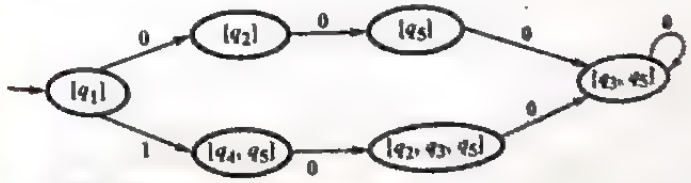



*Fig. 2.46*

***Prob.35. Construct a minimum state automaton equivalent to the automaton given in fig. 2.47.***

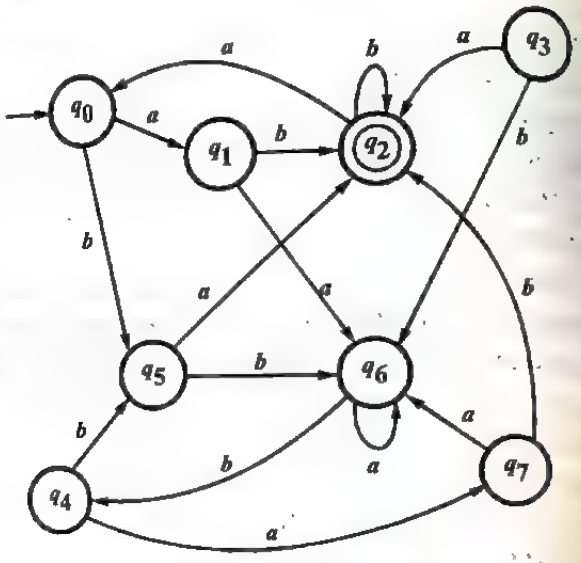



***Fig. 2.47***

***Sol.*** It will be easier if we construct the transition table as shown in table 2.7.

Application of Step 1, which is defined in Q.22, gives –

$$Q_1^0 = F = \{q_2\},\ Q_2^0 = Q - Q_1^0$$

So,

$$\pi_0 = \{\{q_2\},\ \{q_0, q_1, q_3, q_4, q_5, q_6, q_7\}\}$$

If we observe $\pi_0$, we find that $\{q_2\}$ cannot be further partitioned. So $\{q_2\}$. Consider $q_0$ and $q_1 \in Q_2^0$. The entries under $a$-column corresponding to $q_0$ and $q_1$, are $q_1$ and $q_6$; they lie in $Q_2^0$. The entries under $b$-column are $q_5$ and $q_2$. $q_2 \in Q_1^0$ and $q_5 \in Q_2^0$. Therefore, $q_0$ and $q_1$ are not 1-equivalent. In the same way, $q_0$ is not 1-equivalent to $q_3$, $q_5$, $q_7$.

**Table 2.7 Transition**

| State | Input a | |
|---|---|---|
| $\rightarrow q_0$ | $q_1$ | $q_$ |
| $q_1$ | $q_6$ | $q_$ |
| $(q_2)$ | $q_0$ | $q_$ |
| $q_3$ | $q_2$ | $q_$ |
| $q_4$ | $q_7$ | $q_$ |
| $q_5$ | $q_2$ | $q_$ |
| $q_6$ | $q_6$ | $q_$ |
| $q_7$ | $q_6$ | $q_$ |

Now take $q_0$ and $q_4$. The entries under $a$-column are $q_1$, and $q_7$. Both are in $Q_2^0$. The entries under $b$-column are $q_5$, $q_5$. So $q_0$ and $q_4$ are 1-equivalent. Similarly, $q_0$ and $q_6$ are 1-equivalent. $\{q_0, q_4, q_6\}$ is a subset in $\pi_1$. So, $Q_2' = \{q_0, q_4, q_6\}$.

Repeat the construction by considering this time, $q_1$ and any one of states $q_3$, $q_5$ and $q_7$. We find, the $q_1$ is not 1-equivalent to $q_3$ and $q_5$ 1-equivalent to $q_7$. Hence $Q_3' = \{q_1, q_7\}$. The elements left in $Q_2^0$ are $q_3$ and $q_5$. By considering the entries under $a$-column and $b$-column, we find that $q_3$ and $q_5$ are 1-equivalent. So $Q_4' = \{q_3, q_5\}$.

Therefore –

$$\pi_1 = \{\{q_2\},\ \{q_0, q_4, q_6\},\ \{q_1, q_7\},\ \{q_3, q_5\}\}$$

$\{q_2\}$ will be in $\pi_2$ as it cannot be further partitioned. Now, the entries under $a$-column corresponding to $q_0$ and $q_4$ are $q_1$ and $q_7$, and these lie in the same equivalence class in $\pi_1$. The entries under $b$-column are $q_5$, $q_5$. So $q_0$ and $q_4$ are 2-equivalent. But there is no 2-equivalence between $q_0$ and $q_6$. Hence, $\{q_0, q_4, q_6\}$ is partitioned into $\{q_0, q_4\}$ and $\{q_6\}$. $q_1$ and $q_7$ are 2-equivalent. Also, $q_3$ and $q_5$ are 2-equivalent. Thus,

$$\pi_2 = \{\{q_2\},\ \{q_0, q_4\},\ \{q_6\},\ \{q_1, q_7\},\ \{q_3, q_5\}\}$$

$q_0$ and $q_4$ are 3-equivalent. Also $q_1$ and $q_7$ are 3-equivalent. And $q_3$ and $q_5$ are also 3-equivalent. Therefore,

$$\pi_3 = \{\{q_2\},\ \{q_0, q_4\},\ \{q_6\},\ \{q_1, q_7\},\ \{q_3, q_5\}\}$$

As, $\pi_2 = \pi_3$, $\pi_2$ gives the equivalence classes, the minimum state automaton is given as,

$$M_D = \{Q', \{a, b\}, \delta', q_0', F'\}$$

where, $Q' = \{[q_2], [q_0, q_4], [q_6], [q_1, q_7], [q_3, q_5]\}$

$q'_0 = [q_0, q_4]$, $F' = [q_2]$

and $\delta'$ is given by the transition table shown in table 2.8.

**Table 2.8 Transition Table of Minimum State Automaton**

| State | Input a | Input b |
|---|---|---|
| $[q_0,q_4]$ | $[q_1,q_7]$ | $[q_3,q_5]$ |
| $[q_1,q_7]$ | $[q_6]$ | $[q_2]$ |
| $[q_2]$ | $[q_0,q_4]$ | $[q_2]$ |
| $[q_3,q_5]$ | $[q_2]$ | $[q_6]$ |
| $[q_6]$ | $[q_6]$ | $[q_0,q_4]$ |

The transition graph (or diagram) for the minimum state automaton is shown in fig. 2.48.

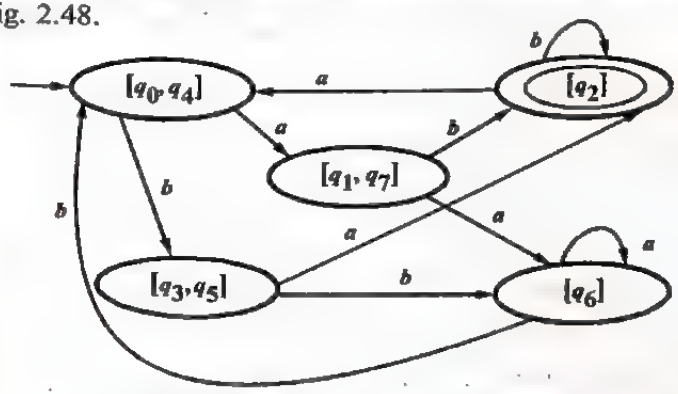



***Fig. 2.48 Minimum State Automaton***

The states $q_0$ and $q_4$ are identified and treated as one state. Also $q_1$, $q_7$ and $q_3$, $q_5$. But the transitions in both the diagrams (i.e., of fig. 2.47 and fig. 2.48) are the same. If there is an arrow from $q_i$ to $q_j$ with label $a$ (or $b$), then

there is an arrow from $[q_i]$ to $[q_j]$ with the same label $a$ (or $b$) in the g for minimum state automaton.

Symbolically, if $\delta(q_i, a) = q_j$, then $\delta'([q_i], a) = [q_j]$.

***Prob.36. Minimize the states of the following finite automata.***

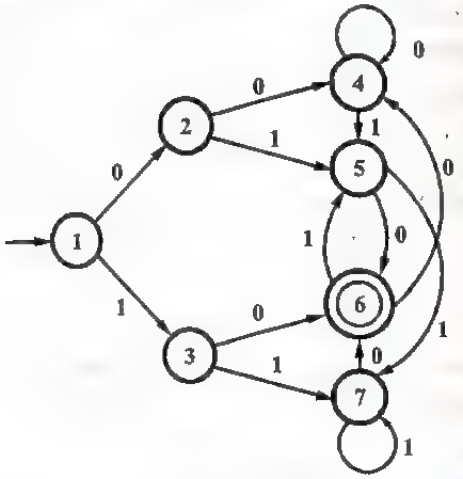



*Fig. 2.49* *(R.G.P.V., Dec.*

***Sol.*** We can construct the transition table shown in table 2.9.

Since there is only one final state 6,

$$Q_1^0 = \{6\},\ Q_2^0 = Q - Q_1^0$$

Hence, $\pi_0 = \{\{6\}, \{1,2,3,4,5,7\}\}$

As $\{6\}$ cannot be partitioned further,

$$Q_1^1 = \{6\}$$

Now 1 is 1-equivalent to 2, 4, but not to 3, 5, 7 and so $Q_2^1 = \{1, 2, 4\}$. 3 is 1-equivalent to 5,7.

Hence, $Q_3^1 = \{3,5,7\}$. Thus

$$\pi_1 = \{\{6\}, \{1,2,4\}, \{3,5,7\}\}$$

$$Q_1^2 = \{6\}$$

As 1 is 2-equivalent to 2,4. So

$$Q_2^2 = \{1,2,4\}$$

As 3 is 2-equivalent to 5,7 So

$$Q_3^2 = \{3,5,7\}$$

Therefore,

$$\pi_2 = \{\{6\}, \{1,2,4\}, \{3,5,7\}\}$$

**Table 2.9 Transition**

| State | Inpu 0 |
|---|---|
| →1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | 4 |
| 5 | 6 |
| (6) | 4 |
| 7 | 6 |

**Table 2.10 Transition Ta Minimum State Automa**

| States | Input 0 | 1 |
|---|---|---|
| →[1,2,4] | [1,2,4] | [3,5 |
| [3,5, 7] | [6] | [3,5 |
| ([6]) | [1,2,4] | [3,5 |



As $\pi_2 = \pi_1$, $\pi_1$ gives us the equivalence classes, the minimum state automaton is –

$M = (Q', \{0, 1\}, \delta', q_0', F')$, where

$Q' = \{[6], [1,2,4], [3,5,7]\}$

$q_0' = [1,2,4]$, $F' = [6]$

$\delta$ is given by table 2.10.

The transition diagram is given in fig. 2.50.

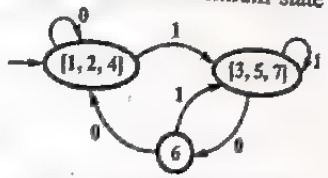



*Fig. 2.50 Minimum State Automaton*

***Prob.37. Construct a minimum state automaton equivalent to given automaton, whose transition graph is as ahead.***

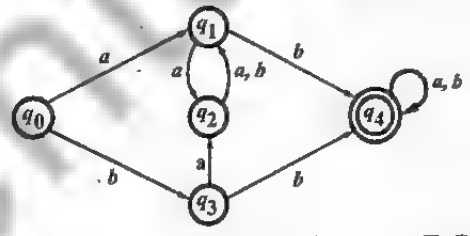



*Fig. 2.51* *(R.G.P.V., June 2008)*

***Sol.*** We construct the transition table in table 2.11.

**Table 2.11 Transition Table**

| State/Σ | a | b |
|---|---|---|
| → $q_0$ | $q_1$ | $q_3$ |
| $q_1$ | $q_2$ | $q_4$ |
| $q_2$ | $q_1$ | $q_1$ |
| $q_3$ | $q_2$ | $q_4$ |
| ($q_4$) | $q_4$ | $q_4$ |

By applying step (i) we get

$$Q_1^0 = F = \{q_4\}$$

$$Q_2^0 = Q - Q_1^0$$

$$= \{q_0, q_1, q_2, q_3\}$$

So, $\pi_0 = \{\{q_4\}, \{q_0, q_1, q_2, q_3\}\}$

As $\{q_4\}$ cannot be partitioned further,

$$Q'_1 = \{q_4\}$$

Now $q_0$ is 1-equivalent to $q_2$ but not to $q_1$, $q_3$ and so $Q'_2 = \{q_0, q_2\}$. $q_1$ is 1-equivalent to $q_3$. Hence $Q'_3 = \{q_1, q_3\}$. Thus,

$$\pi_1 = \{\{q_4\}, \{q_0, q_2\}, \{q_1, q_3\}$$

$$Q_1^2 = \{q_4\}$$

$q_0$ is 2-equivalent to $q_2$

$$Q_2^2 = \{q_0, q_2\}$$

$q_1$ is 2-equivalent to $q_3$

$$Q_3^2 = \{q_1, q_3\}$$

Thus, $$\pi_2 = \{\{q_4\}, \{q_0, q_2\}, \{q_1, q_3\}\}$$

As $\pi_2 = \pi_1$, $\pi_2$ gives us equivalence classes, the minimum state automaton is $M = (Q', \{a, b\}, \delta', q'_0, F')$, where

$$Q' = \{\{q_4\}, \{q_0, q_2\}, \{q_1, q_3\}\}$$
$$q'_0 = \{q_0, q_2\}$$
$$F' = \{q_4\}$$

$\delta'$ is given by table 2.12.

**Table 2.12**

| State/Σ | a | b |
|---|---|---|
| $[q_0, q_2]$ | $[q_1, q_3]$ | $[q_1, q_3]$ |
| $[q_1, q_3]$ | $[q_0, q_2]$ | $[q_4]$ |
| $[q_4]$ | $[q_4]$ | $[q_4]$ |

***Prob.38. Construct a minimum state automaton for the following:***

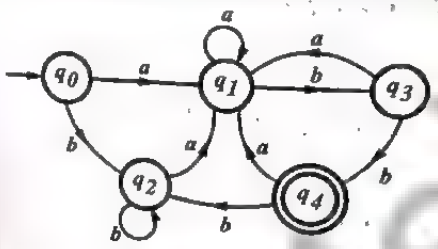



*Fig. 2.52* ***(R.G.P.V., June***

***Sol.*** We can construct the transition table shown in table 2.13.

As there is only one final state, i.e., $q_4$

$$Q_1^0 = \{q_4\}, Q_2^0 = Q - Q_1^0 = \{q_0, q_1, q_2, q_3\}$$

Hence, $$\pi_0 = \{\{q_4\}, \{q_0, q_1, q_2, q_3\}\}$$

As $\{q_4\}$ cannot be further partitioned

$$Q_1' = \{q_4\}$$

Now $q_0$ is 1-equivalent to $q_0$, $q_1$, $q_2$ but not to $q_3$, so

$$Q_2' = \{q_0, q_1, q_2\}$$

$q_3$ cannot be further partitioned, so

$$Q_3' = \{q_3\}$$

**Table 2.13 Transition**

| State | Input | |
|---|---|---|
| | a | |
| → $q_0$ | $q_1$ | |
| $q_1$ | $q_1$ | |
| $q_2$ | $q_1$ | |
| $q_3$ | $q_1$ | |
| (($q_4$)) | $q_1$ | |



Thus, $$\pi_1 = \{\{q_4\}, \{q_0, q_1, q_2\}, \{q_3\}\}$$

$$Q_1^2 = \{q_4\}$$

$q_0$ is 2-equivalent to $q_1$ but not to $q_2$, so

$$Q_2^2 = \{q_0, q_2\}$$
$$Q_3^2 = \{q_1\}$$
$$Q_4^2 = \{q_3\}$$

Thus, $$\pi_2 = \{\{q_4\}, \{q_0, q_2\}, \{q_1\}, \{q_3\}\}$$

$$Q_1^3 = \{q_4\}$$

Now, $q_0$ is 3-equivalent to $q_2$, so

$$Q_2^3 = \{q_0, q_2\}$$
$$Q_3^3 = \{q_1\}$$
$$Q_4^3 = \{q_3\}$$

Thus, $$\pi_3 = \{\{q_4\}, \{q_0, q_2\}, \{q_1\}, \{q_3\}\}$$

Since $\pi_3 = \pi_2$, $\pi_2$ gives us the equivalence classes, the minimum state automaton is –

$$M_D = (Q', \{a, b\}, \delta', q_0', F')$$

where, $Q' = \{[q_4], [q_0, q_2], [q_1], [q_3]\}$

$q_0' = [q_0, q_2]$, $F' = [q_4]$ and $\delta'$ is given by table 2.14.

The transition diagram for $M_D$ is given in fig. 2.53.

**Table 2.14 Transition Table for Minimum State Automaton**

| State | Input | |
|---|---|---|
| | a | b |
| $[q_0, q_2]$ | $[q_1]$ | $[q_0, q_2]$ |
| $[q_1]$ | $[q_1]$ | $[q_3]$ |
| $[q_3]$ | $[q_1]$ | $[q_4]$ |
| $[q_4]$ | $[q_1]$ | $[q_0, q_2]$ |

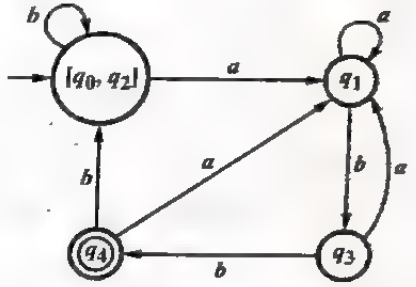



*Fig. 2.53 Minimum State Automaton*

***Prob.39. Construct a minimum state automation equivalent to a given automation M whose transition table is defined below –***

| State | Input | |
|---|---|---|
| | a | b |
| → $q_0$ | $q_0$ | $q_3$ |
| $q_1$ | $q_2$ | $q_5$ |
| $q_2$ | $q_3$ | $q_4$ |
| $q_3$ | $q_0$ | $q_5$ |
| $q_4$ | $q_0$ | $q_6$ |
| $q_5$ | $q_1$ | $q_4$ |
| (($q_6$)) | $q_1$ | $q_3$ |

***(R.G.P.V., Dec. 2017)***

***Sol.*** As there is only one final state i.e. $q_6$

$\therefore \quad Q_1^0 = \{q_6\}, \quad Q_2^0 = Q - Q_1^0 = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

Hence $\pi_0 = \{\{q_6\}, \{q_0, q_1, q_2, q_3, q_4, q_5\}\}$

As $q_6$ cannot be further partitioned, so

$$Q_1^1 = \{q_6\}$$

Now $q_0$, is 1-equivalent to $q_1$, $q_2$, $q_3$, and $q_5$ but not to $q_4$, so

$$Q_2^1 = \{q_0, q_1, q_2, q_3, q_5\} \text{ and } Q_3^1 = \{q_4\}$$

Hence, $\pi_1 = \{\{q_6\}, \{q_0, q_1, q_2, q_3, q_5\}, \{q_4\}\}$

Again $q_6$ and $q_4$ cannot be further partitioned, so

$$Q_1^2 = \{q_6\}, \quad Q_2^2 = \{q_4\}$$

Now $q_0$ is 2-equivalent to $q_1$, $q_3$ but not $q_2$, $q_5$ so

$$Q_3^2 = \{q_0, q_1, q_3\}, \quad Q_4^2 = \{q_2, q_5\}$$

Hence, $\pi_2 = \{\{q_6\}, \{q_4\}, \{q_0, q_1, q_3\}, \{q_2, q_5\}\}$

Again, $q_6$ and $q_4$ cannot be further partitioned, so

$$Q_1^3 = \{q_6\}, \quad Q_2^3 = \{q_4\}$$

Now $q_0$, is not 3-equivalent to $q_1$ and $q_3$, so

$$Q_3^3 = \{q_0, q_1, q_3\}$$

Again $q_2$ is 3-equivalent to $q_5$, so

$$Q_4^3 = \{q_2, q_5\}$$

Hence $\pi_3 = \{\{q_6\}, \{q_4\}, \{q_0, q_1, q_3\}, \{q_2, q_5\}\}$

Since, $\pi_3 = \pi_2$, therefore minimum state automation is

$$M_D = (Q', \{a, b\}, \delta', q'_0, F')$$

where, $Q' = \{[q_0, q_1, q_3], [q_2, q_5], [q_4], [q_6]\}$

$q'_0 = [q_0, q_1, q_3]$, $F' = [q_6]$ and $\delta'$ is given by the table 2.15.

**Table 2.15**

| State | Input | |
|---|---|---|
| | *a* | *b* |
| $[q_0, q_1, q_3]$ | $[q_0, q_1, q_3]$ | $[q_0, q_1, q_3]$ |
| $[q_2, q_5]$ | $[q_0, q_1, q_3]$ | $[q_4]$ |
| $[q_4]$ | $[q_0, q_1, q_3]$ | $[q_6]$ |
| $[q_6]$ | $[q_0, q_1, q_3]$ | $[q_0, q_1, q_3]$ |

# UNIT 3

## INTRODUCTION OF CONTEXT-FREE GRAMMAR – DERIVATION TREES, AMBIGUITY, SIMPLIFICATION OF CFGs, NORMAL FORMS OF CFGs – CHOMSKY NORMAL FORM AND GREIBACH NORMAL FORMS

***Q.1. What do you understand by a context-free grammar ? Give some examples.***

***Or***

***Define context-free grammar.*** ***(R.G.P.V., Dec. 2007)***

***Or***

***Define CFG.*** ***(R.G.P.V., Dec. 2016)***

***Ans.*** A grammar $G = (V, T, P, S)$ is said to be context-free, if every production in P is of the form –

$$A \to x$$

where, $A \in V$ and $x \in (V \cup T)^*$.

A language is said to be context-free if there is a context-free grammar $G$ such that $L = L(G)$.

Every regular grammar is context-free, so a regular language is also context-free. But as we know that there are non-regular languages.

We can say that, the family of regular languages is a proper subset of the family of context-free languages.

For example –

The grammar $G = (\{S\}, \{a,b\}, P, S)$

having productions

$$S \to aSa$$
$$S \to bSb$$
$$S \to \lambda$$

is context-free. A typical derivation in this grammar is as follows –

$$S \Rightarrow aSa \Rightarrow aaSaa \Rightarrow aabSbaa \Rightarrow aabbaa$$

***Q.2. Why CFG is not considered adequate for describing n language ? Explain with suitable example.*** ***(R.G.P.V., Dec.***

***Ans.*** For a number of reasons, context free grammars are not in g regarded as adequate for the description of natural languages like Englis example if we extend the productions

$$\begin{aligned} &< \text{sentence} > \rightarrow < \text{noun phrase} > < \text{verb phrase} > \\ &< \text{noun phrase} > \rightarrow < \text{adjective} > < \text{noun phrase} > \\ &< \text{noun phrase} > \rightarrow < \text{noun} > \\ &< \text{noun} > \rightarrow \text{boy} \\ &< \text{adjective} > \rightarrow \text{little} \end{aligned}$$

to encompass all of English, we would be able to derive "rock" as a phrase and "runs" as a verb phrase. Thus "rock runs" would be a sen which is nonsense. Clearly some semantic information is necessary to re meaningless strings that are made to associate the meaning of the sen with its derivation.

***Q.3. How many types of grammar are there ? Explain each of t*** ***(R.G.P.V., Dec.***

***Or***

***Write short note on Chomsky classification of grammar.*** ***(R.G.P.V., June***

***Or***

***Write short note on Chomsky hierarchy.*** ***(R.G.P.V., June***

***Or***

***Explain the types of the grammar.*** ***(R.G.P.V., June)***

***Or***

***Explain Chomsky classification of languages with suitable exam*** ***(R.G.P.V., Nov.***

***Ans.*** Chomsky classified the grammars into four types in ter productions (types 0-3).

A type 0 grammar is any phrase structure grammar without any restric

A production without any restrictions is called a type 0 productio

A production of the form $\phi A \psi \rightarrow \phi \alpha \psi$ is called a type 1 produc $\alpha \neq \Lambda$. In type 1 productions erasing of $A$ is not permitted. Here $A$ variable, $\phi$ is called the left context $\psi$, the right context, and $\phi\alpha\psi$ the replace string.

A grammar is called type 1 or context-sensitive or context dependen its productions are type 1 productions. The production $S \rightarrow \Lambda$ is also al in a type 1 grammar, but in this case $S$ does not appear on the right-hand of any production. The language generated by a type 1 grammar is ca type 1 or context-sensitive language.



A type 2 production is a production of the form $A \rightarrow \alpha$, where $A \in V_N$ and $\alpha \in (V_N \cup \Sigma)^*$. That is, the L.H.S. has no left context or right context. For example, $S \rightarrow Aa$, $A \rightarrow a$, $B \rightarrow abc$, $A \rightarrow \Lambda$ are type 2 productions.

A grammar is called a type 2 grammar if it contains only type 2 productions. It is also called a context-free grammar. A language generated by a context-free grammar is called type 2 language or a context-free language.

A production of the form $A \rightarrow a$ or $A \rightarrow aB$, where $A, B \in V_N$ and $a \in \Sigma$, is called a type 3 production.

A grammar is called a type 3 or regular grammar if all its productions are type 3 productions. A productions $S \rightarrow \Lambda$ is allowed in type 3 grammar, but in this case $S$ does not appear on the right-hand side of any production.

***Q.4. Describe context free and context sensitive grammar.*** ***(R.G.P.V., June 2015)***

***Ans.*** Refer to Q.3.

***Q.5. What do you mean by context free grammar ? Explain the types of grammar.*** ***(R.G.P.V., June 2011)***

***Ans.*** Refer to Q.1 and Q.3.

***Q.6. Define derivation tree or parse tree with an example.***

***Ans.*** A derivation tree is an ordered tree in which nodes are labelled with the left sides of productions, also in which, the children of a node represent its corresponding right sides.

For example, let $G = (\{S, A\}, \{a, b\}, P, S)$, where $P$ consists of

$S \rightarrow aAS \mid a \mid SS$, $A \rightarrow SbA \mid ba$.

Fig. 3.1 shows a derivation tree representing

$S \Rightarrow SS \Rightarrow aS \Rightarrow aaAS \Rightarrow aabaS \Rightarrow aabaa$

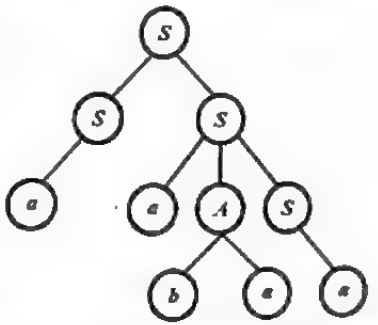



***Fig 3.1 Derivation Tree***

***Q.7. What are leftmost and rightmost derivations ? Explain with suitable example.*** ***(R.G.P.V., Dec. 2015)***

***Or***

***Explain LMD and RMD with example.*** ***(R.G.P.V., June 2011)***

***Ans.*** A derivation is said to be ***leftmost,*** if in each step the leftmost vari in the sentential form is replaced. And if in each step the rightmost variabl replaced, the derivation is called ***rightmost.***

For example, given a grammar having productions

$$S \to aAB,$$
$$A \to bBb,$$
$$B \to A|\Lambda.$$

Then,

$S \Rightarrow aAB \Rightarrow abBbB \Rightarrow abAbB \Rightarrow abbBbbB \Rightarrow abbbbB \Leftarrow ab$

is a leftmost derivation of the string *abbbb*. The rightmost derivation of same string *abbbb* is

$S \Rightarrow aAB \Rightarrow aA \Rightarrow abBb \Rightarrow abAb \Rightarrow abbBbb \Rightarrow abbbb.$

***Q.8. Explain leftmost derivation, rightmost derivation and parse tree with suitable example.*** ***(R.G.P.V., June 20***

***Or***

***Define leftmost derivation, rightmost derivation and parse tree with suitable example.*** ***(R.G.P.V., Nov. 20***

***Ans.*** Refer to Q.7 and Q.6.

***Q.9. Explain ambiguity in context-free grammars.***

***Or***

***Write short note on ambiguity in grammar.*** ***(R.G.P.V., June 20***

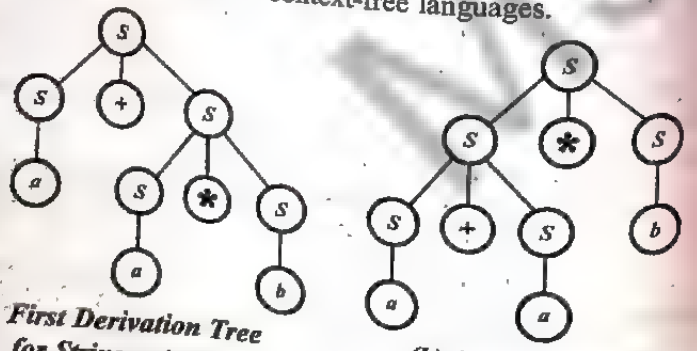
***Or***

***Explain ambiguous grammar problem.*** ***(R.G.P.V., June 20***

***Or***

***Define ambiguity.*** ***(R.G.P.V., Dec. 20***

***Ans.*** Sometimes there may occur some ambiguous sentences in a language we are using. Let us take an example of the following english sentence – "In books selected information is given". The word 'selected' may refer books or information. So the sentence may be parsed in two different ways. The same situation may arise in context-free languages.




***(a) First Derivation Tree for String $a + a * b$***




***(b) Second Derivation Tree for String $a + a * b$***

***Fig. 3.2 Two Different Derivation Trees for One String $a + a * b$***



The same terminal string may be the yield of two different derivation trees. So there may be two different leftmost derivations for a string. This leads to the definition of ambiguous sentences in a context-free language, as follows –

A terminal string $u \in L(G)$ is ambiguous, if there exist two or more derivation trees for $u$ (or there exist two or more leftmost derivations of $u$).

For example, $G = (\{S\}, \{a, b, +, *\}, P, S)$, where $P$ consists of

$$S \to S + S \mid S * S \mid a \mid b$$

We have two derivation trees for string $a + a * b$, given in fig. 3.2.

The leftmost derivations of $a + a * b$ induced by the two derivation trees are as follows –

(i) $S \Rightarrow S + S \Rightarrow a + S \Rightarrow a + S * S \Rightarrow a + a * S \Rightarrow a + a * b$

(ii) $S \Rightarrow S * S \Rightarrow S + S * S \Rightarrow a + S * S \Rightarrow a + a * S \Rightarrow a + a * b$

Therefore, string $a + a * b$ is ambiguous.

So, we can define that a context-free grammer $G$ is ambiguous if there exists some $u \in L(G)$, which is ambiguous.

***Q.10. Explain rule for simplification of CFG*** ***(R.G.P.V., June 2010)***

***Ans.*** In a context-free grammar $G$, consists of $(V, T, P, S)$, it may not be necessary to use all the symbols in $V \cup T$, or all the production in $P$ for deriving sentences. So when we study a context-free language $L(G)$, we try to eliminate symbols and productions in $G$ which are not useful for derivation of sentences. For example, consider

$$G = (\{S, A, B, C, E\}, \{a, b, c\}, P, S)$$

where $P = \{S \to AB, A \to a, B \to b, B \to C, E \to c|\Lambda\}$

It is easy to observe that $L(G) = \{ab\}$. Let

$$G' = (\{S, A, B\}, \{a, b\}, P', S),$$

where $P' = \{S \to AB, A \to a, B \to b\}$ and $L(G) = L(G')$

We have eliminated the symbols $C$, $E$ and $c$ and the productions $B \to C$, $E \to c|\Lambda$. The following points regarding the symbols and productions can be eliminated –

(i) $C$ does not derive any terminal string

(ii) $E$ and $c$ do not appear in any sentential form

(iii) $E \to \Lambda$ is a null production

(iv) $B \to C$ simply replaces $B$ by $C$.

In this way one can restrict the format of productions without reducing the generative power of context-free grammars.

***Q.11. Explain the following –***

***(i) Parse tree (ii) Useless symbols.*** ***(R.G.P.V., Dec. 2007)***

*Ans.* ***(i) Parse Tree*** – Refer to Q.6.

***(ii) Useless Symbols*** – A symbol that is not useful is useless sym
Clearly, if a variable is either not live or not reachable, then it is useless.

Let $G = (V, T, P, S)$ be a grammar. A symbol $X$ is useful if there
derivation $S \overset{*}{\Rightarrow} \alpha \times \beta \overset{*}{\Rightarrow} w$ for some α, β and $w$, where w is in $T^*$. Other
$X$ is useless symbol. There are two aspects to usefulness.

(a) Some terminal string must be derivable from $X$ and

(b) $X$ must occur in some string derivable from $S$.

These two conditions are not, sufficient to guarantee that $X$ is use
since $X$ may occur only in sentential forms that contain a variable from wh
no terminal string can be derived.

***Q.12. Define the normal forms for context-free grammars.***

***Or***

***Define CNF and GNF.*** ***(R.G.P.V., June 20***

***Or***

***Define the following –***

***(i) Chomsky normal form***

***(ii) Greibach normal form.***

***(R.G.P.V., Dec. 2002, 20***

*Ans.* In a context-free grammar, the R.H.S. of a production can be
string of variables and terminals. When the productions in $G$ satisfy cert
restrictions, then $G$ is said to be in a ***'normal form'***. Among several 'norm
forms', we explain two 'normal forms', i.e., Chomsky normal form
Greibach normal form.

***(i) Chomsky Normal Form (CNF)*** – In the Chomsky normal fo
(CNF), we have restriction on the length of R.H.S. and the nature of symb
in the R.H.S. of productions.

A context-free grammar is in ***Chomsky normal form*** if all productio
are of the form –

$$A \to BC \quad \text{or} \quad A \to a$$

where $A$, $B$, $C$, are in $V$ and $a$ is in $T$.

For example – the grammar

$$S \to AS \mid a$$
$$A \to SA \mid b$$

is in Chomsky normal form. While the grammar

$$S \to AS \mid AAS$$
$$A \to SA \mid aa$$

is not in Chomsky normal form because both productions $S \to AAS$ a
$A \to aa$ violate the conditions of the definition, given above.



We can also say that –

"Any context-free grammar $G$ with $\Lambda \notin L(G)$ has an equivalent grammar $G'$ in Chomsky normal form."

***(ii) Greibach Normal Form (GNF)*** – Greibach normal form (GNF) is another normal form quite useful in some proofs and constructions. A context-free grammar generating the set accepted by a pushdown automaton is in Greibach normal form.

Here, restrictions are put, not on the length of the right sides of a production, but on the positions of the terminals and variables. Arguments which justify Greibach normal form are little complicated and not very transparent. Also, constructing a grammar in Greibach normal form equivalent to any context-free grammar is tedious. Nevertheless, Greibach normal form has many theoretical and practical consequences.

A context-free grammar is said to be in Greibach normal form if all of its productions have the following form –

$$A \to aA_1$$

where $a \in T$ and $A_1 \in V^*$.

For example, the following grammar –

$$S \to AB$$
$$A \to aA|bB|b,$$
$$B \to b$$

is not in Greibach normal form. But, using the substitution, we immediately get the equivalent grammar as follows –

$$S \to aAB|bBB|bB.$$
$$A \to aA \mid bB \mid b.$$
$$B \to b$$

This is in Greibach normal form and we can also say that –

"For every context-free grammar $G$, with $\Lambda \notin L(G)$, there exists an equivalent grammar $G'$ in Greibach normal form".

***Q.13. Write short note on CNF.*** ***(R.G.P.V., Dec. 2004)***

***Or***

***Write short note on Chomsky normal form.*** ***(R.G.P.V., June 2003)***

*Ans.* Refer to Q.12 (i).

***Q.14. What are steps to convert the context-free grammar to Chomsky normal form ? Explain each step with suitable examples. (R.G.P.V., June 2004)***

*Ans.* As we know for every context-free grammar, there is an equivalent grammar in Chomsky normal form. Following procedure gives the method of constructing a grammar in CNF equivalent to a context-free grammar –

***(i) Elimination of Null Productions and Unit Productions.*** apply theorem to eliminate null productions. We then apply theorem to resulting grammar to eliminate chain productions. Let the grammar thus obtained be $G = (V, T, P, S)$.

***(ii) Elimination of Terminals on R.H.S.*** – We define $G_1 = (V_1, P_1, S)$, where $P_1$ and $V_1$ are constructed as follows –

(a) All the productions in $P$ of the form $A \to a$ or $A \to BC$ included in $P_1$. All the variables in $V$ are included in $V_1$.

(b) Consider $A \to X_1X_2.....X_n$, with some terminal on R.H.S. If $X_i$ is a terminal, say $a_i$, add a new variable $C_{ai}$ to $V'$ and $C_{ai} \to a_i$ to $P_1$. production $A \to X_1X_2 .....X_n$ every terminal on R.H.S. is replaced by corresponding new variable and the variables on the R.H.S. are retained. resulting production is added to $P_1$. Thus we get $G_1 = (V_1, T, P_1, S)$.

***(iii) Restricting the Number of Variables on R.H.S.*** – For production in $P_1$, the R.H.S. consists of either a single terminal (or $\Lambda$ in $S \to \Lambda$) or two or more variables. We define $G_2 = (V_2, T, P_2, S)$ as follows –

(a) All productions in $P_1$ are added to $P_2$ if they are in required form. All the variables in $V_1$ are added to $V_2$.

(b) Consider $A \to A_1A_2.....A_m$, where $m \geq 3$. We introduce new productions $A \to A_1C_1$, $C_1 \to A_2C_2$,....., $C_{m-2} \to A_{m-1}A_m$, and variables $C_1, C_2, ....., C_{m-2}$. These are added to $P_2$ and $V_2$, respectively.

Thus we get $G_2$ in Chomsky normal form.

***Q.15. Write an algorithm to convert a grammar to Greibach normal form (GNF).***

***(R.G.P.V., June 20***

***Or***

***Explain GNF conversion steps.*** ***(R.G.P.V., June 20***

***Ans.*** Following procedure gives the method of constructing a gram in GNF equivalent to a context-free grammar.

**Construction of $G$** – (i) We eliminate null productions and then constr a grammar $G$ in Chomsky normal form. We renames the variables as $A_1$, ....., $A_n$ with $S = A_1$. We write $G$ as $(\{A_1, A_2, ....., A_n\}, T, P, A_1)$.

(ii) To get the productions in the form $A_i \to a\gamma$ or $A_i \to A_j\gamma$, wh $j > i$, convert the $A_i$-productions $(i = 1, 2, ....., n-1)$ to the fo $A_i \to A_j\gamma$, such that $j > i$. By the principle of induction, the construction ca carried out for $i = 1, 2, ....., n$. Thus for $i = 1, 2, ....., n-1$, $A_i$-production is of the form $A_i \to A_j\gamma$, where $j > i$ or $A_i \to a\gamma'$. $A_n$-production is of the form $A_n \to A_n\gamma$ or $A_n \to a\gamma'$.

(iii) Convert $A_n$-productions to the form $A_n \to a\gamma$. Here, productio of the form $A_n \to A_n\gamma$ are eliminated using lemma 2. The result $A_n$-productions are of the form $A_n \to a\gamma$.



(iv) Modify $A_i$-productions to the form $A_i \to a\gamma$ for $i = 1, 2,..., n-1$. At the of step (iii), the $A_n$-productions are of the form $A_n \to a\gamma$. The $A_{n-1}$ productions are of the form $A_{n-1} \to a\gamma'$ or $A_{n-1} \to A_n\gamma$. By applying lemma 1 we eliminate productions of the form $A_{n-1} \to A_n\gamma$. The resulting $A_{n-1}$-productions are in the required form. We repeat the construction by considering $A_{n-2}$, $A_{n-3}$,...., $A_1$.

(v) Modify $Z_i$-productions. Every time we apply lemma 2, we get a new variable (we take it as $Z_i$ when we apply the lemma for $A_i$-productions.) The $Z_i$-productions are of the form $Z_i \to \alpha Z_i$ or $Z_i \to \alpha$ (where $\alpha$ is obtained from $A_i \to A_i\alpha$), and hence of the form $Z_i \to a\gamma$ or $Z_i \to A_k\gamma$ for some $k$. At the end of step (iv), the R.H.S. of any $A_k$-production starts with a terminal. So we can apply lemma 1 to eliminate $Z_i \to A_k\gamma$. Thus at the end of step (v), we get an equivalent grammar $G_1$ in GNF.

## NUMERICAL PROBLEMS

***Prob.1. Consider the following productions –***

$$S \to aAS \mid a$$
$$A \to SbA \mid SS \mid ba$$

***Find out LMD, RMD, Parse tree for string aabbaa.***

***(R.G.P.V., June 2006)***

***Sol.*** **LMD (Leftmost Derivation) –**

$$S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aabAS$$
$$\Rightarrow aabbaS \Rightarrow aabbaa$$

**RMD (Rightmost Derivation) –**

$$S \Rightarrow aAS \Rightarrow aAa \Rightarrow aSbAa$$
$$\Rightarrow aSbbaa \Rightarrow aabbaa$$

**Parse Tree** – The parse tree is shown in fig. 3.3.

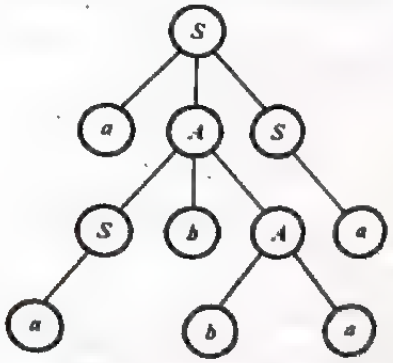



***Fig. 3.3 Parse Tree***

***Prob.2. Let G be the grammar –***

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

***For the string aaabbabbba find –***

***(i) Leftmost derivation***

***(ii) Rightmost derivation***

***(iii) Parse tree***

***(iv) Is the grammar unambiguous.***

***(R.G.P.V., June 20***

***Or***

***Let G be the grammar.***

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

***For the string aaabbabbba find a***

***(i) Leftmost derivation***

***(ii) Rightmost Derivation***

***(iii) Parse tree.***

***(R.G.P.V., Dec. 20***

***Or***

***Let G be the grammar –***

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

***For the string aaabbabbba, find leftmost and rightmost derivation***

***(R.G.P.V., Dec. 20***

***Or***

***Consider the following CFG –***

$S \rightarrow aB \mid bA,\ A \rightarrow a|aS|bAA,\ B \rightarrow b|bS|aBB$

***For string "aaabbabbba", find –***

***(i) Leftmost derivation (ii) Rightmost derivation (iii) Parse***

***(R.G.P.V., June 20***

***Sol. (i) Leftmost Derivation –***

$S \Rightarrow aB \Rightarrow aaBB \Rightarrow aaaBBB \Rightarrow aaabSBB \Rightarrow aaabbABB \Rightarrow aaabb$

$\Rightarrow aaabbabB \Rightarrow aaabbabbS \Rightarrow aaabbabbbA \Rightarrow aaabbabbba$

***(ii) Rightmost Derivation –***

$S \Rightarrow aB \Rightarrow aaBB \Rightarrow aaBbS \Rightarrow aaBbbA \Rightarrow aaBbba \Rightarrow aaaBB$

$\Rightarrow aaaBbbba \Rightarrow aaabSbbba \Rightarrow aaabbAbbba \Rightarrow aaabbabbba$

***(iii) Parse Tree*** – The parse tree or derivation tree is given in fig.



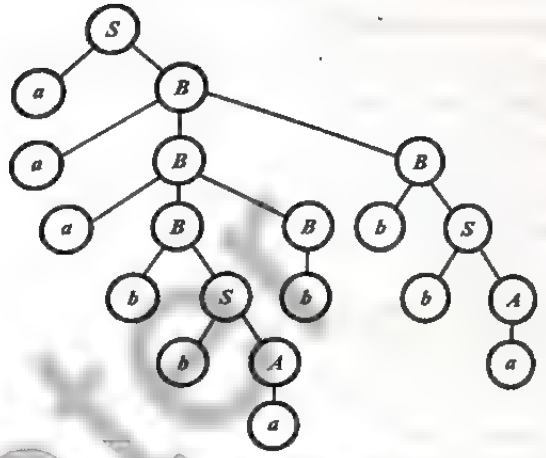
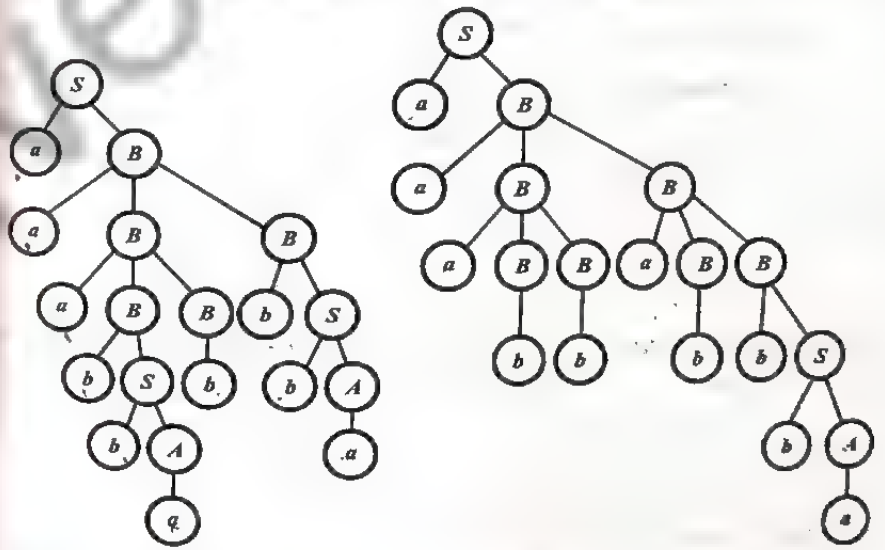


***Fig. 3.4 Derivation Tree***

***(iv) Ambiguity*** – Here, we have more than one parse tree for the given string *aaabbabbba* as shown in fig. 3.5. The string has two different derivations as shown in fig. 3.5 (a) and 3.5 (b), respectively,



***(a)*** ***(b)***

***Fig. 3.5 Derivation Trees 'for aaabbabbba***

$S \Rightarrow aB \Rightarrow aaBB \Rightarrow aaaBBB \Rightarrow aaabSBB \Rightarrow aaabbABB$

$\Rightarrow aaabbaBB \Rightarrow aaabbabB \Rightarrow aaabbabbS \Rightarrow aaabbabbbA$

$\Rightarrow aaabbabbba$

and $S \Rightarrow aB \Rightarrow aaBB \Rightarrow aaaBBB \Rightarrow aaabBB \Rightarrow aaabbB \Rightarrow aaabbaBB$

$\Rightarrow aaabbabB \Rightarrow aaabbabbS \Rightarrow aaabbabbbA \Rightarrow aaabbabbba$

***So, the grammar is ambiguous.***

***Prob.3. Let G be the grammar***

$$S \to 0B \mid 1A,\ A \to 0 \mid 0S \mid 1AA,\ B \to 1 \mid 1S \mid 0BB$$

***For the string 00110101 find –***

***(i) LMD***

***(ii) RMD***

***(iii) Parse tree.***

***(R.G.P.V., Dec. 20***

***Sol. (i) Leftmost Derivation –***

$S \Rightarrow 0B \Rightarrow 00BB \Rightarrow 001B \Rightarrow 0011S$
$\Rightarrow 00110B \Rightarrow 001101S \Rightarrow 0011010B$
$\Rightarrow 00110101$

***(ii) Rightmost Derivation –***

$S \Rightarrow 0B \Rightarrow 00BB \Rightarrow 00B1S \Rightarrow 00B10B$
$\Rightarrow 00B10S \Rightarrow 00B1010B \Rightarrow 00B10101$
$\Rightarrow 00110101$

***(iii) Parse Tree –*** The derivation tree is given in fig. 3.6.

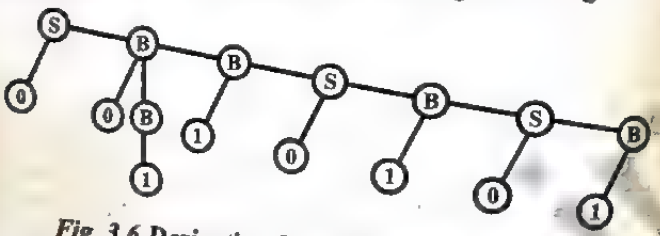


***Fig. 3.6 Derivation Tree for String 00110101***

***Prob.4. Derive the string "aabbabba" for leftmost derivation & rightmost derivation using a CFG given by –***

$S \to aB \mid bA$
$A \to a \mid aS \mid bAA$
$B \to b \mid bS \mid aBB$

***And also draw the derivation tree.***

***(R.G.P.V., Dec. 20***

***Sol.*** **Leftmost Derivation –**

$S \Rightarrow aB \Rightarrow aaBB \Rightarrow aabB \Rightarrow aabbS \Rightarrow aabbaB \Rightarrow aabbabS \Rightarrow aabbabbA \Rightarrow aabbabba$

**Rightmost Derivation –**

$S \Rightarrow aB \Rightarrow aaBB \Rightarrow aaBbS \Rightarrow aaBbbA \Rightarrow aaBbba \Rightarrow aabSbba \Rightarrow aabbAbba \Rightarrow aabbabba$

**Parse Tree –** The parse tree or derivation tree is given in fig. 3.7.



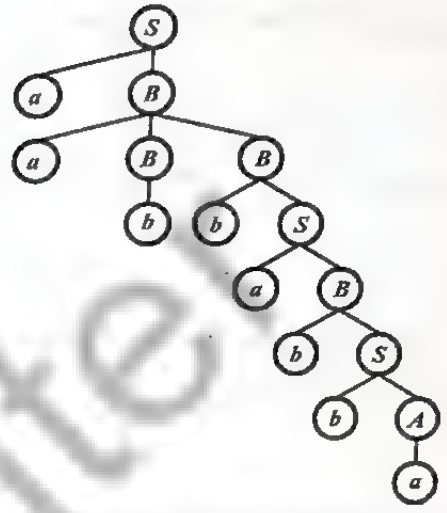


***Fig. 3.7 Derivation Tree***

***Prob.5. Consider the grammar –***

$$S \to aS|aSbS|\in$$

***This grammar is ambiguous. Show in particular that the string aab has two –***

***(i) Parse trees***

***(ii) Leftmost derivations***

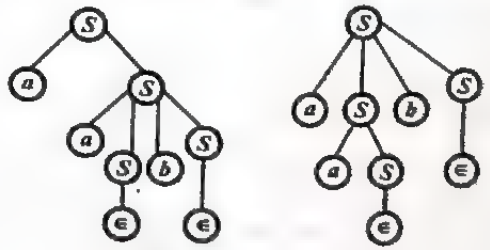
***(iii) Rightmost derivations.***

***(R.G.P.V., Dec. 2008)***

***Sol. (i) Parse Tree –*** Given that

$S \to aS|aSbS|\in$

This grammar is ambiguous. So that string "aab" can be generated from this grammar in two distinct ways, as shown in the following derivation trees or parse trees in fig. 3.8.



***(a)*** ***(b)***

***Fig. 3.8 Parse Trees***

| | |
|---|---|
| $S \to aS$ | $S \to aSbS$ |
| $S \to aaSbS$ | $S \to aaSbS$ |
| $S \to aab$ | $S \to aab$ |

***(ii) Leftmost Derivations*** – "*aab*" has two distinct leftmost derivations –

$$S \Rightarrow aS \Rightarrow aaSbS \Rightarrow aab$$
$$S \Rightarrow aSbS \Rightarrow aaSbS \Rightarrow aab$$

***(iii) Rightmost Derivations*** – Also "*aab*" has two distinct rightmost derivations –

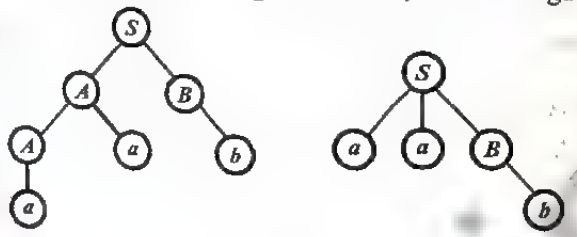
$$S \Rightarrow aS \Rightarrow aaSbS \Rightarrow aab$$
$$S \rightarrow aSbS \Rightarrow aSb \Rightarrow aaSb \Rightarrow aab$$

***Prob.6. Show that the following grammar is ambiguous –***

$$S \rightarrow AB \mid aaB$$
$$A \rightarrow a \mid Aa$$
$$B \rightarrow b$$

***(R.G.P.V., Dec. 20***

***Sol.*** To prove that grammar $G$ is ambiguous, we have to find a string $u \in L(G)$, which is ambiguous. Consider $u = aab \in L(G)$. Then we get two derivation trees for $u$, as shown in fig. 3.9. Thus, $G$ is ambiguous.



***(a)*** ***(b)***

***Fig. 3.9 Derivation Trees for aab***

***Prob.7. Prove that the given grammer is ambiguous.***

***(i)*** $S \rightarrow aSb|SS|\in$

***(ii)*** $S \rightarrow AB|aaB,\ A \rightarrow a|Aa,\ B \rightarrow b$

***(R.G.P.V., Dec. 2***

***Sol. (i)*** $S \rightarrow aSb|SS|\in$ – To prove that the given grammer $G$ is ambiguous, we have to find a string $u \in L(G)$, which is ambiguous. Consider $u = aabb \in L(G)$. Then we get two derivation tree for $u$, as shown in fig. 3.10. Thus $G$ is ambiguous.

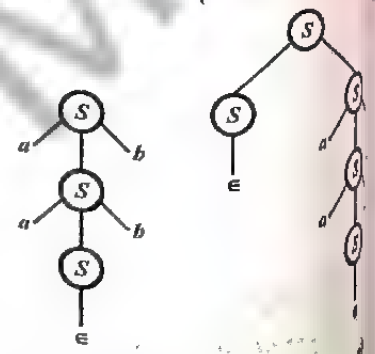


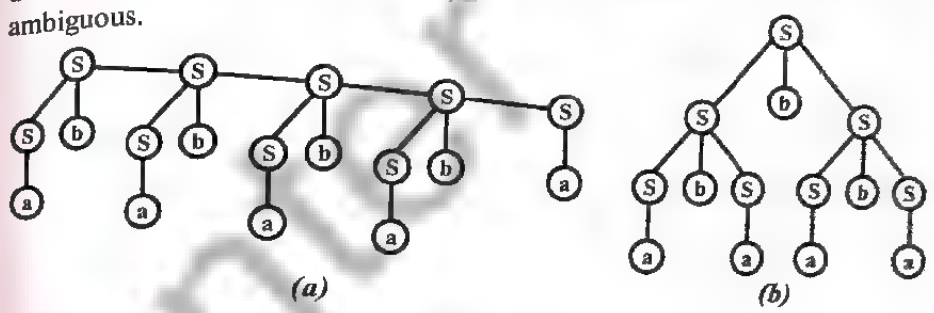
***Fig. 3.10 Derivation Tree for a***

***(ii)*** $S \rightarrow AB|aaB,\ A \rightarrow a|Aa,\ B \rightarrow b$ – Refer to Prob.6.



***Prob.8. If G is the grammar*** $S \rightarrow SbS \mid a$, ***show that G is ambiguous.***
***(R.G.P.V., Dec. 2017)***

***Sol.*** To prove that grammar G is ambiguous, we have to find a string u ∈ L(G), which is ambiguous. Consider u = abababa ∈ L(G). Then we get two derivation trees for u as shown in fig. 3.11 (a) and fig. 3.11 (b). Thus G is ambiguous.

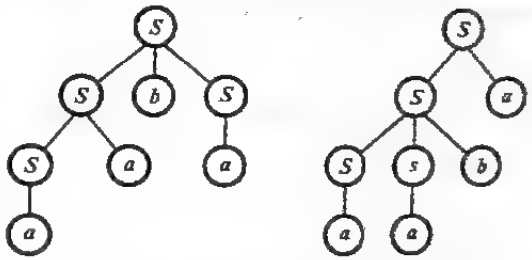


***(a)*** ***(b)***

***Fig. 3.11***

***Prob.9. Show that the grammar*** $S \rightarrow a|Sa|bSS|SSb|SbS$ ***is ambiguous.***
***(R.G.P.V., June 2011, 2015)***

***Sol.*** To prove that grammar $G$ is ambiguous we have to find a string $u \in L(G)$, which is ambiguous. Consider $u = aaba \in L(G)$. Then we get two derivation trees for $u$ (see fig. 3.12). Thus, $G$ is ambiguous.



***(a)*** ***(b)***

***Fig. 3.12 Derivation Trees for aaba***

***Prob.10. Show that the grammar*** $S \rightarrow a|abSb|aAb,\ A \rightarrow bS \mid aAAb$ ***is ambiguous.***
***(R.G.P.V., Dec. 2014)***

***Sol.*** To prove that grammar $G$ is ambiguous we have to find a string $u \in L(G)$, which is ambiguous. Consider $u = abab \in L(G)$. Then we get two derivation trees for $u$ as shown in fig. 3.13. Thus, $G$ is ambiguous.

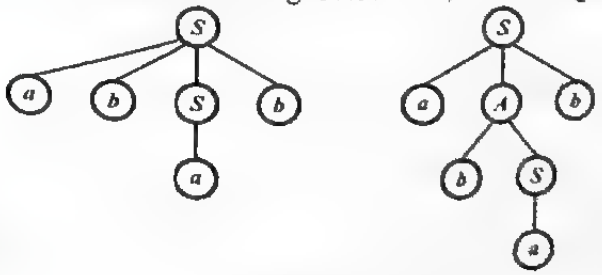


***Fig. 3.13***

***Prob.11. Show that the following grammar is ambiguous***

$$S \rightarrow aSbS|bSaS|\in$$ ***(R.G.P.V., Dec. 20**

***Sol.*** To prove that grammar $G$ is ambiguous, we have to find a string $u$ $L(G)$, which is ambiguous. Consider $u = abab \in L(G)$. Then, we get t derivation trees for $u$, as shown in fig. 3.14. Thus, $G$ is ambiguous.

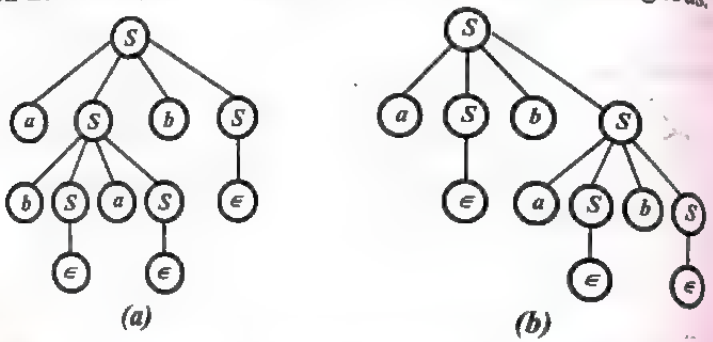



***Fig. 3.14 Derivation Trees for abab***

***Prob.12. Briefly explain what is meant by ambiguity in CFG ? Sh that the following grammar is ambiguous –***

$$E \rightarrow E + E|E^* E|2|3|4$$ ***(R.G.P.V., June 20***

***Or***

***What is ambiguity in grammar ? Show that the given grammar ambiguous.***

$$E \rightarrow E + E|E^* E|2|3|4$$ ***(R.G.P.V., Nov. 20***

***Sol.*** **Ambiguity in CFG** – Refer to Q.9.

Now, consider the grammar

$G = (V, T, E, P)$ with

$V = \{E\}$, $T = \{2, 3, 4, +, *\}$,

and productions

$$E \rightarrow E + E$$
$$E \rightarrow E * E$$
$$E \rightarrow 2|3|4.$$

The strings $(a + b) * c$ and $a * b + c$ are in $L(G)$. It is easy to see that t grammar generates a restricted subset of arithmetic expressions for FORTR and Pascal-like programming languages. The grammar is ambiguous. F instance, the string $a + b * c$ has two different derivation trees, as shown fig. 3.15.

One way to resolve the ambiguity, as is done in programming manuals, associate precedence rules with the operators + and *. Since * has high precedence over + we would take fig. 3.15 (a) as the correct parsing s indicates that b*c is a subexpression to be evaluated before performing addition. It is better to rewrite the grammar so that only parsing is possib



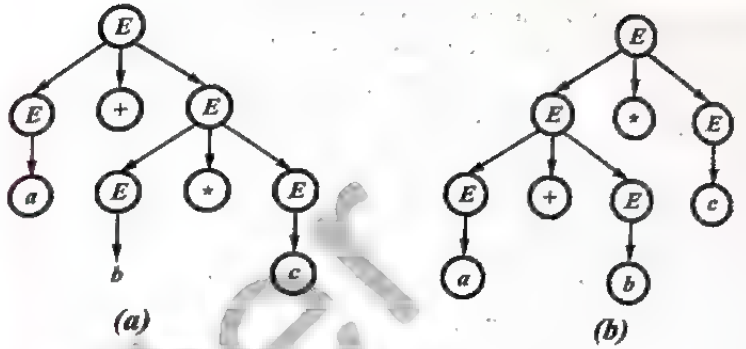



***Fig. 3.15 Two Derivation Tree for a + b * c***

***Prob.13. Show that the following grammar is ambiguous –***

$$E \rightarrow E + E|E^*E|2|3|4$$

***(R.G.P.V., June 2010)***

***Sol.*** Refer to Prob.12.

***Prob.14. What do you mean by ambiguity ? Show that the grammar***

$$S \rightarrow aB \mid ab$$
$$A \rightarrow a$$
$$B \rightarrow C \mid b$$
$$C \rightarrow D$$
$$D \rightarrow E$$
$$E \rightarrow a$$

***is ambiguous.*** ***(R.G.P.V., Dec. 2010)***

***Sol.*** **Ambiguity** – Refer to Q.9.

**Problem** – The given grammar $G$ is –

$G = (\{S, A, B, C, D, E\}, \{a, b\}, P, S)$

where, $P$ consists of

$$S \rightarrow aB \mid ab$$
$$A \rightarrow a$$
$$B \rightarrow C \mid b$$
$$C \rightarrow D$$
$$D \rightarrow E$$
$$E \rightarrow a$$

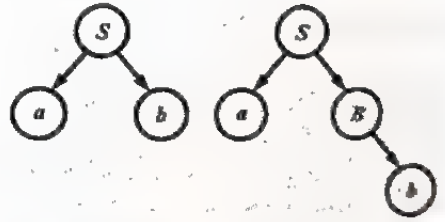



***Fig. 3.16 Derivation Trees for ab***

We have two derivation trees for $ab$ given in fig. 3.16. Therefore, the grammar is ambiguous.

***Prob.15. Prove that the CFG G1 with productions –***

$$S1 \rightarrow S1 + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (S1) \mid a$$

***is unambiguous.*** ***(R.G.P.V., Dec. 2005)***

**Sol. Proof** – We wish to show that every string $x$ in $L(G1)$ has only one leftmost derivation from $S1$. The proof will be by mathematical induction on $|x|$, and it will actually be easier to prove something apparently stronger – that any $x$ derivable from one of the variables $S1$, $T$, or $F$, $x$ has only one leftmost derivation from that variable.

For the basis step, we observe that $a$ can be derived from any of the three variables, and that in each case there is only one derivation.

In the induction step, we assume that $k \geq 1$ and that for every $y$ derivable from $S1$, $T$, or $F$ for which $|y| \leq k$, y has only one leftmost derivation from that variable. We wish to show the same result for a string $x$ with $|x| = k+1$.

Consider first the case in which $x$ contains at least one + not within parentheses. Because the only +'s in strings derivable from $T$ or $F$ are within parentheses, $x$ can be derived only from $S1$, and any derivation of $x$ must begin $S1 \Rightarrow S1 + T$, where this + is the last + in $x$ that is not within parentheses. Therefore, any leftmost derivation of $x$ from $S1$ has the from

$$S1 \Rightarrow S1 + T \Rightarrow^* y + T \Rightarrow^* y + z$$

where the last two steps represent leftmost derivation of $x$ from $S1$ and from $T$, respectively, and the + is still the last one not within parentheses. The induction hypothesis tells us that $y$ has only one leftmost derivation from $S1$ and $z$ has only one from $T$. Therefore, $x$ has only one leftmost derivation from $S1$.

Next consider the case in which $x$ contains no + outside parentheses but at least one * outside parentheses. This time $x$ can be derived only from $T$; any derivation from $S1$ must begin $S1 \Rightarrow T \Rightarrow T * F$, and any derivation from $T$ must begin $T \Rightarrow T * F$. In either case, the * must be the last one that is not within parentheses. As in the first case, the subsequent steps of the leftmost derivation must be

$$T * F \Rightarrow^* y * F \Rightarrow^* y * z$$

consisting first of a leftmost derivation of $y$ from $T$ and then of a leftmost derivation of $z$ from $F$. Again, the induction hypothesis tell us that there is only one possible way for these derivations to proceed, and therefore there is only one leftmost derivation of $x$ from $S1$ or $T$.

Finally, suppose $x$ contains no +'s or *'s outside parentheses. Then $x$ can be derived from any of the variables, however, the only derivation from $T$ begins $S1 \Rightarrow T \Rightarrow F \Rightarrow (S1)$, and the only derivation from $T$ or $F$ begins the same way, with the first one or two steps omitted. Therefore, $x = (y)$, with $S1 \Rightarrow^* y$. By the induction hypothesis, $y$ has only one leftmost derivation from $S1$, and it follows that $x$ has only one from each of the three variables. This completes the proof.



***Prob.16. Check whether the given grammar is ambiguous or not –***

$$S \rightarrow iC + S$$
$$S \rightarrow iC + ScS$$
$$S \rightarrow a$$
$$C \rightarrow b$$

***(R.G.P.V., Dec. 2011)***

***Sol.*** To check whether the given grammar $G$ is ambiguous, we have to find a string $u \in L(G)$, which is ambiguous. Consider $u = ib + ib + aca$. Then, we get two derivation trees for $u$ (see fig. 3.17). Thus, $G$ is ambiguous.

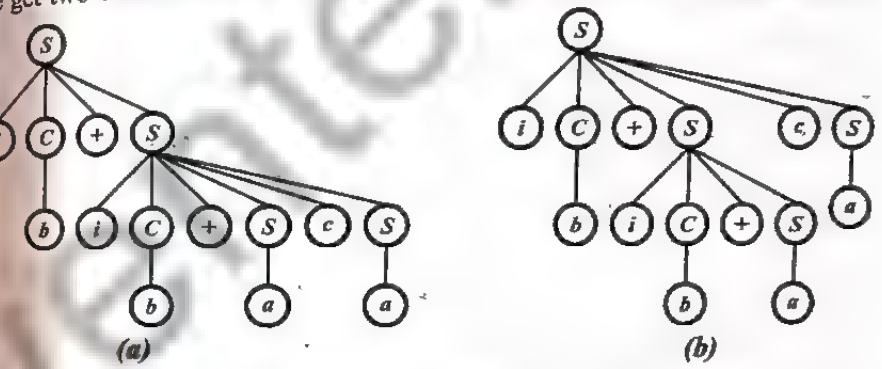



***Fig. 3.17 Derivation Trees for ib + ib + aca***

***Prob.17. Remove null production from the grammar –***

$$S \rightarrow ABAB$$
$$A \rightarrow 0A \mid \in$$
$$B \rightarrow 1B \mid 1$$

***(R.G.P.V., June 2007)***

***Sol.*** We see that the null production is $A$. Then we construct the CFG without null production in the following manner –

$$S \rightarrow ABAB \mid ABB \mid BAB \mid BB$$
$$A \rightarrow 0A \mid 0$$
$$B \rightarrow 1B \mid 1$$

***Prob.18. Construct the reduced grammar equivalent to grammar –***

$$S \rightarrow aAa$$
$$A \rightarrow Sb \mid bCC \mid DaA$$
$$C \rightarrow abb \mid DD$$
$$E \rightarrow aC$$
$$D \rightarrow aDA$$

***(R.G.P.V., Dec. 2014)***

***Sol.*** **Step 1 –**

$$U_1 = \{C\} \text{ as } C \rightarrow abb$$

the only production with a terminal string on the R.H.S.

$$U_2 = \{C\} \cup \{E, A\}$$

as $E \to aC$ and $A \to bCC$ are productions with R.H.S. in $(T \cup \{\})$

$U_3 = \{C, E, A\} \cup \{S\}$ as $S \to aAa$ and $aAa$ is in $(T \cup U_2)$

$U_4 = U_3 \cup \phi$

Hence,

$V' = U_3 = \{S, A, C, E\}$

$P' = \{A_1 \to \alpha | \alpha \in (V' \cup T)^*\}$

$= \{S \to aAa,\ A \to Sb | bCC,\ C \to abb,\ E \to aC\}$

$G_1 = (V', \{a,b\}, S, P')$

**Step 2 –**

$U_1 = \{S\}$

As we have $S \to aAa$, $U_2 = \{S\} \cup \{A, a\}$

As $A \to Sb|bCC$, $U_3 = \{S, A, a\} \cup \{S, b, C\} = \{S, A, C, a, b\}$

As we have $C \to abb$, $U_4 = U_3 \cup \{a, b\} = U_3$

Hence,

$P'' = \{A_1 \to \alpha | A_1 \in U_3\} = \{S \to aAa,\ A \to Sb|bCC,\ C \to$

Therefore,

$G' = (\{S, A, C\}, \{a, b\}, S, P'')$

***is the reduced grammar.***

***Prob.19. Eliminate $\in$-productions from a given CFG –***

$S \to ABCBCDA$

$A \to CD$

$B \to Cb$

$C \to a \mid \in$

$D \to bD \mid \in$

***(R.G.P.V., Dec.***

***Sol.*** First remove $\in$-production from $C$ and $D$. After removing production from $C$ and $D$, the resultant $CFG$ is

$S \to ABCBCDA|ABBCDA|ABCBDA|ABBDA|ABCBCA|AB$
$ABCBA|ABBA$

$A \to CD|C|D|\in$

$B \to Cb|b$

$C \to a$

$D \to bD|b$

Now, remove $\in$-production from $A$

$S \to ABCBCDA|ABBCDA|ABCBDA|ABBDA|ABCBCA|AB$
$ABCBA|ABBA|BCBCDA|ABCBCD|BCBCD|BBCDA|$
$|BBCD|BCBDA|ABCBD|BCBD|BBDA|ABBD|BBD|$
$|ABCBC|BCBC|BBCA|ABBC|BBC|BCBA|ABCD|BCB|BBA|$



$A \to CD|C|D$

$B \to Cb|b$

$C \to a$

$D \to bD|b$

**This is the required grammar without $\in$-production.**

***Prob.20. Remove all unit-productions from –***

$S \to Aa|B,\ B \to A|bb,\ A \to a|bc|B$

***Sol.*** The dependency graph for the unit productions is given as in fig. 3.18.

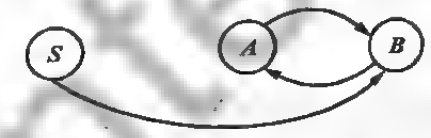



***Fig. 3.18***

We observe from it that $S \overset{*}{\Rightarrow} A$, $S \overset{*}{\Rightarrow} B$, $B \overset{*}{\Rightarrow} A$, and $A \overset{*}{\Rightarrow} B$. Hence, we dd to the original non-unit productions –

$S \to Aa,$

$A \to a|bc,$

$B \to bb,$

he new rules –

$S \to a|bc|bb,\ A \to bb,\ B \to a|bc,$

o get the original grammar as follows –

$S \to a|bc|bb|Aa,\ A \to a|bb|bc,\ B \to a|bb|bc.$

***This is the required grammar without unit productions.***

***Prob.21. Find a CFG with no useless symbols equivalent to –***

$S \to AB|CA,$ $B \to BC|AB$

$A \to a,$ $C \to aB|b$

***(R.G.P.V., June 2003)***

***Sol.*** **Step 1 –**

$U_1 = \{A, C\}$ as $A \to a$ and $C \to b$ are productions with a terminal tring on R.H.S.

$U_2 = \{A, C\} \cup \{A_1 | A_1 \to \alpha$ with $\alpha \in (T \cup \{A, C\})^*\}$

$= \{A, C\} \cup \{S\}$ as we have $S \to CA$

$U_3 = \{A, C, S\} \cup \{A_1 | A_1 \to \alpha$ with $\alpha \in (T \cup \{S, A, C\})^*\}$

$= \{A, C, S\} \cup \phi$

As $U_3 = U_2$

$V' = U_2 = \{S, A, C\}$

$P' = \{A_1 \to \alpha \mid A_1\alpha \in (V' \cup T)^*\} = \{S \to CA,\ A \to a, C \to b\}$

Thus,

$$G_1 = (\{S,A,C\}, \{a,b\}, S, \{S \to CA, A \to a, C \to b\}).$$

**Step 2 –**

$$U_1 = \{S\}$$

As we have production $S \to CA$ and $S \in U_1$, $U_2 = \{S\} \cup \{A,C\}$

As $A \to a$ and $C \to b$ are productions with $A, C \in U_2, U_3 = \{S,A,C\}$

As $U_3 = V' \cup T, P'' = \{A_1 \to \alpha | A_1 \in U_3\} = P$

Therefore,

$$G' = (\{S,A,C\},\{a,b\},S,\{S \to CA, A \to a, C \to b\})$$

*is the reduced grammar.*

***Prob.22. What do you mean by useless production ? Consider the g***

***G = (V, T, P, S) where V, T, P, S are given as –***

***V = {S, A, B, C, E}***

***T = {a, b, c}***

***S = {S} and***

***P consists of***

$S \to AB$

$A \to a$

$B \to b$

$B \to C$

$E \to c$

***Eliminate useless symbols and productions from the grammar.*** ***(R.G.P.V., Dec.***

***Sol.*** A production rule not affecting the language is called u production. One invariably wants to remove productions from a gramm can never take part in any derivation. For example, in the gramma entire production set is

$$S \to aSb \mid \lambda \mid A$$
$$A \to aA$$

the production $S \to A$ clearly plays no role, as a cannot be transforme terminal string. While $A$ can occur in a string derived from $S$, this ca lead to a sentence. Removing this production leaves the language un and is a simplification by any definition.

Identify the production that lead to terminal string in the given p i.e.,

$$S \to AB$$
$$A \to a$$
$$B \to b$$
$$E \to c$$



$B \to C$ does not lead to a terminal string, so remove this production.

Eliminate the variables that cannot be reachable from the start symbols. Hence the variable $E$ can be removed as it is not possible to reach $E$ from $S$. Thus, the resulting productions take the form

$$S \to AB$$
$$A \to a$$
$$B \to b$$

***Prob.23. Simplify the given grammar –***

$$S \to aSB \mid aA \mid bB$$
$$A \to aA \mid \in$$
$$B \to bB \mid \in$$

***(R.G.P.V., June 2009)***

***Sol.*** The nullable productions are –

$$A \to \in \text{ and } B \to \in$$

To eliminate $A \to \in$ from the above grammar, we replace $A$ on the right side of the productions.

$$S \to aA$$
$$A \to aA$$

Replace each occurrence of $A$ by $\in$ in each of these productions to obtain the non $\in$-productions, to be added to the grammar, the list of these productions is –

$$S \to a$$
$$A \to a$$

Add these productions to the grammar and eliminate $A \to \in$ from the grammar to get the following grammar –

$$S \to aSB \mid aA \mid bB \mid a$$
$$A \to aA \mid a$$
$$B \to bB \mid \in$$

To eliminate $B \to \in$ from the grammar, the non $\in$-productions to be added are obtained as follows –

List of the productions containing $B$ on right side is –

$$S \to aSB \mid bB$$
$$B \to bB$$

Replace each occurrence of $B$ by $\in$ is each of these productions to obtain the non $\in$-productions, to be added to the grammar, the list of these productions is –

$$S \to aS \mid b$$
$$B \to b$$

Add these productions to the grammar and eliminate $B \to \in$ from the grammar to obtain the following grammar –

$$S \to aSB \mid aA \mid bB \mid a \mid aS \mid b$$
$$A \to aA \mid a$$
$$B \to bB \mid b$$

Thus,

$G_1 = (\{S,A,C\}, \{a,b\}, S, \{S \to CA, A \to a, C \to b\})$.

**Step 2 –**

$U_1 = \{S\}$

As we have production $S \to CA$ and $S \in U_1$, $U_2 = \{S\} \cup \{A,C\}$

As $A \to a$ and $C \to b$ are productions with $A, C \in U_2, U_3 = \{S,A,$

As $U_3 = V' \cup T, P'' = \{A_1 \to \alpha | A_1 \in U_3\} = P'$

Therefore,

$G' = (\{S,A,C\},\{a,b\},S,\{S \to CA, A \to a, C \to b\})$

***is the reduced grammar.***

***Prob.22. What do you mean by useless production ? Consider the g***

***G = (V, T, P, S) where V, T, P, S are given as –***

***V = {S, A, B, C, E}***

***T = {a, b, c}***

***S = {S} and***

***P consists of***

$S \to AB$

$A \to a$

$B \to b$

$B \to C$

$E \to c$

***Eliminate useless symbols and productions from the grammar.*** ***(R.G.P.V., De***

***Sol.*** A production rule not affecting the language is called production. One invariably wants to remove productions from a gram can never take part in any derivation. For example, in the gramma entire production set is

$S \to aSb \mid \lambda \mid A$

$A \to aA$

the production $S \to A$ clearly plays no role, as a cannot be transform terminal string. While $A$ can occur in a string derived from $S$, this ca lead to a sentence. Removing this production leaves the language un and is a simplification by any definition.

Identify the production that lead to terminal string in the given i.e.,

$S \to AB$

$A \to a$

$B \to b$

$E \to c$



$B \to C$ does not lead to a terminal string, so remove this production.

Eliminate the variables that cannot be reachable from the start symbols. Hence the variable $E$ can be removed as it is not possible to reach $E$ from $S$. Thus, the resulting productions take the form

$S \to AB$

$A \to a$

$B \to b$

***Prob.23. Simplify the given grammar –***

$S \to aSB \mid aA \mid bB$

$A \to aA \mid \in$

$B \to bB \mid \in$ ***(R.G.P.V., June 2009)***

***Sol.*** The nullable productions are –

$A \to \in$ and $B \to \in$

To eliminate $A \to \in$ from the above grammar, we replace $A$ on the right side of the productions.

$S \to aA$

$A \to aA$

Replace each occurrence of $A$ by $\in$ in each of these productions to obtain the non $\in$-productions, to be added to the grammar, the list of these productions is –

$S \to a$

$A \to a$

Add these productions to the grammar and eliminate $A \to \in$ from the grammar to get the following grammar –

$S \to aSB \mid aA \mid bB \mid a$

$A \to aA \mid a$

$B \to bB \mid \in$

To eliminate $B \to \in$ from the grammar, the non $\in$-productions to be added are obtained as follows –

List of the productions containing $B$ on right side is

$S \to aSB \mid bB$

$B \to bB$

Replace each occurrence of $B$ by $\in$ is each of these productions to obtain the non $\in$-productions, to be added to the grammar, the list of these productions is –

$S \to aS \mid b$

$B \to b$

Add these productions to the grammar and eliminate $B \to \in$ from the grammar to obtain the following grammar –

$S \to aSB \mid aA \mid bB \mid a \mid aS \mid b$

$A \to aA \mid a$

$B \to bB \mid b$

***Prob.24. Remove the ∈-production from the following CFG.***

$$S \to XYX$$
$$X \to 0X \mid \in$$
$$Y \to 1Y \mid \in$$

***(R.G.P.V., June***

***Sol.*** We find that the nullable variables are $A$, $B$, $C$. Then, we co... the grammar in the following way –

$$S \to XYX \mid YX \mid XX \mid XY \mid X \mid Y$$
$$X \to 0X \mid 0$$
$$Y \to 1Y \mid 1$$

***Prob.25. Eliminate unit, useless and ∈-productions from the gr...***

$$S \to aA \mid aBB,\ A \to aaA \mid \in,\ B \to bB \mid bbC,\ C \to B.$$

***(R.G.P.V., Dec. ...***

***Sol.*** Grammar is

$$S \to aA \mid aBB$$
$$A \to aaA \mid \in$$
$$B \to bB \mid bbC$$
$$C \to B$$

***(i) Elimination of Unit Production***

Since $C \to B$ is unit production

So replace $B$ with bB/bbC in $C \to B$

i.e.,

$$S \to aA \mid aBB$$
$$A \to aaA \mid \in$$
$$B \to bB \mid bbC$$
$$C \to bB \mid bbC$$

***(ii) Elimination of Useless Production***

Here, $B \to bB \mid bbC$ and

$C \to bB \mid bbC$ both are making an infinite loop (i.e. these ... will never ever end).

So after removing these equations and equation that contains ... variable the reduced grammar will be

$$S \to aA$$
$$A \to aaA \mid \in$$

***(iii) Elimination of ∈-production***

Placing $\in$ in place of $A$ in RHS of every equation which contains ...

$$S \to aA \mid a$$
$$A \to aaA \mid aa$$



Hence, after elimination of unit, useless and ∈-production, the reduced ...rammar is –

$$S \to aA \mid a$$
$$A \to aaA \mid aa$$

***Prob.26. Convert the following CFG into an equivalent CFG in Chomsky ...ormal form –***

$$A \to BAB \mid B \mid \in$$
$$B \to 00 \mid \in$$

***(R.G.P.V., Dec. 2007)***

***Sol.*** Given CFG is –

$$A \to BAB \mid B \mid \in$$
$$B \to 00 \mid \in$$

Let us go through each step of conversion to Chomsky normal form as –

**Step 1** – Elimination of ∈-production.

$$A \to BAB \mid B \mid AB \mid BB \mid BA$$
$$B \to 00$$

**Step 2** – Eliminating unit productions. Here we may simply add the ...roduction $A \to 00$ and delete $A \to B$.

**Step 3** – Restricting the right sides of productions to single terminals or ...tring of two or more variables.

$$A \to ABA \mid AB \mid BB \mid BA \mid X_0X_0$$
$$B \to X_0X_0$$
$$X_0 \to 0$$

The final Chomsky normal form is as

$$A \to AT_1 \qquad T_1 \to BA$$
$$A \to AB \mid BB \mid BA \mid X_0X_0$$
$$B \to X_0X_0$$
$$X_0 \to 0$$

***Prob.27. Convert the following CFG to CNF***

$$S \to ABA$$
$$A \to aA \mid \in$$
$$B \to bB \mid \in$$

***(R.G.P.V., Dec. 2012, 2014, June 2015)***

***Sol.*** **Step 1** – Write all the productions separately

$$S \to ABA$$
$$A \to aA$$
$$A \to \in$$
$$B \to bB$$
$$B \to \in$$

**Step 2** – Elimination of ∈ productions

$S \to ABA|BA|AA|AB|A|B$

$A \to aA|a$

$B \to bB|b$

**Step 3** – Elimination of unit productions

$S \to ABA|BA|AA|AB|aA|bB|a|b$

$A \to aA|a$

$B \to bB|b$

**Step 4** – Restricting the right sides of productions to single terminal or string of two or more variables.

$S \to ABA|BA|AA|AB|C_aA|C_bB|a|b$

$A \to C_aA|a$

$B \to C_bB|b$

$C_a \to a$

$C_b \to b$

The final Chomsky normal form is as

$S \to AT_1|AA|AB|BA|C_aA|C_bB|a|b$

$T_1 \to BA$

$A \to C_aA|a$

$B \to C_bB|b$

$C_a \to a$

$C_b \to b$

***Prob.28. Find a grammar in CNF for the grammar G = {N, T, P, S} where N = {S}, T = {a, b} and P = {S → aSb, S → ab} of language $\{a^nb^n \mid n \geq 1\}$.***

***Sol.*** As there are no null productions or unit productions, so there is no need of step 1. Hence, we can proceed to step 2.

**Step 2** – Let $G_1 = (V_N, \{a, b\}, P_1, S)$, where $P_1$ and $V_N$ are constructed as follows –

(i) $S \to aSb, S \to ab$ yields $S \to C_aSC_b, S \to C_aC_b, C_a \to a, C_b$

$V_N = \{S, C_a, C_b\}$

**Step 3** – $P_1$ consists of $S \to C_aSC_b$, $S \to C_aC_b$, $C_a \to a$, $C_b \to$

$S \to C_aSC_b$ is replaced by $S \to C_aC_1$, $C_1 \to SC_b$.

The remaining productions in $P_1$ are added to $P_2$. Let $G_2 = (\{S, C_a, C_b, C_1\}, \{a, b\}, P_2, S\}$

where $P_2$ consists of $S \to C_aC_1$, $C_1 \to SC_b$, $S \to C_aC_b$, $C_a \to a, C_b$

$G_2$ is in CNF and equivalent to the given grammar.



***Prob.29. Reduce the following grammar G to CNF. G is –***

$S \to aAD$

$A \to aB \mid bAB$

$B \to b$

$D \to d$

***Sol.*** As there are no null productions or unit productions, so there is no need to step 1. Hence we can proceed to step 2.

**Step 2** – Let $G_1 = (V_N, \{a, b, d\}, P_1, S)$, where $P_1$ and $V_N$ are constructed as follows –

(i) $B \to b$, $D \to d$ are included in $P_1$.

(ii) $S \to aAD$ gives rise to $S \to C_aAD$ and $C_a \to a$.

$A \to aB$ gives rise to $A \to C_aB$.

$A \to bAB$ gives rise to $A \to C_bAB$ and $C_b \to b$.

$V_N = \{S, A, B, D, C_a, C_b\}$.

**Step 3** – $P_1$ consists of $S \to C_aAD, A \to C_aB \mid C_bAB, B \to b, D \to d, C_a \to a, C_b \to b$.

$A \to C_aB$, $B \to b$, $D \to d$, $C_a \to a$, $C_b \to b$ are added to $P_2$.

$S \to C_aAD$ is replaced by $S \to C_aC_1$ and $C_1 \to AD$.

$A \to C_bAB$ is replaced by $A \to C_bC_2$ and $C_2 \to AB$.

Let, $G_2 = (\{S, A, B, D, C_a, C_b, C_1, C_2\}, \{a, b, d\}, P_2, S)$

where $P_2$ consists of $S \to C_aC_1$, $A \to C_aB \mid C_bC_2$, $C_1 \to AD$, $C_2 \to AB$, $B \to b, D \to d, C_a \to a, C_b \to b$. $G_2$ is in CNF and equivalent to $G$.

***Prob.30. Convert the following CFG into CNF –***

$S \to \sim S \mid [S \supset S] \mid p \mid q$ ***(R.G.P.V., June 2010)***

***Sol.*** Since the given grammar has no unit or null productions, we omit step 1 and proceed to step 2.

**Step 2** – Let $G_1 = (V_N, \Sigma, P_1, S)$, where $P_1$ and $V_N$ are constructed as follows –

(i) $S \to p|q$ are added to $P_1$.

(ii) $S \to \sim S$ induces $S \to AS$ and $A \to \sim$

(iii) $S \to [S \supset S]$ induces $S \to BSCSD$, $B \to [$, $C \to \supset$, $D \to ]$

$V_N = \{S, A, B, C, D\}$

**Step 3** – $P_1$ consists of $S \to p|q$, $S \to AS$, $A \to \sim$, $B \to [$, $C \to \supset$, $D \to ]$, $S \to BSCSD$

$S \to BSCSD$ is replaced by $S \to BC_1$, $C_1 \to SC_2$, $C_2 \to CC_3$, $C_3 \to SD$.

Let

$G_2 = (\{S, A, B, C, D, C_1, C_2, C_3\}, T, P_2, S)$

where $P_2$ consists of $S \to p|q|AS|BC_1$, $A \to \sim$, $B \to [$, $C \to \supset$, $D \to$ $SC_2$, $C_2 \to CC_3$, $C_3 \to SD$. $G_2$ is in CNF and equivalent to the given gr

***Prob.31. What do you mean by normal forms ? Reduce the gra with following productions to CNF.***

$S \to ASA|bA$

$A \to B|S$

$B \to c$

***(R.G.P.V., Dec***

***Sol.*** Refer to Q.12.

Write all the productions separately

$$S \to ASA$$
$$S \to bA$$
$$A \to B$$
$$A \to S$$
$$B \to c$$

The only production $B \to c$ in Chomsky normal form

There is no null production.

Eliminating unit production.

$$S \to ASA \mid bA$$
$$A \to c \mid ASA \mid bA$$
$$B \to c$$

Eliminating useless production

$$S \to ASA \mid bA$$
$$A \to c \mid ASA \mid bA$$

Restricting the right sides of productions to single terminals or str two or more variables. Thus, replace terminal by only right by new v

$B_b \to b$

$$S \to ASA \mid B_bA$$
$$A \to c \mid ASA \mid B_bA$$
$$B_b \to b$$

Replace a combination of two variables on R.H.S. by a single v unit to get the R.H.S. of the productions in two variables.

$$S \to AD_1|B_b A$$
$$A \to c|AD_1|B_b A$$
$$B_b \to b$$
$$D_1 \to SA$$

The grammar is in CNF.



***Prob.32. Convert the following CFG to Chomsky normal form –***

$S \to AACD$

$A \to aAb \mid \in$

$C \to aC \mid a$

$D \to aDa \mid bDb \mid \in$

***(R.G.P.V., Dec. 2005)***

***Sol.*** **Step 1** – Write all the productions separately

$$S \to AACD$$
$$A \to aAb$$
$$A \to \in$$
$$C \to aC$$
$$C \to a$$
$$D \to aDa$$
$$D \to bDb$$
$$D \to \in$$

The only production $C \to a$ in Chomsky normal form.

**Step 2** – Eliminating $\in$-productions –

$$S \to AACD|ACD|AAC|CD|AC|C$$
$$A \to aAb|ab$$
$$C \to aC|a$$
$$D \to aDa|bDb|aa|bb$$

$S \to C$

**Step 3** – Eliminating unit productions –

Thus, the grammar

$$S \to AACD|ACD|AAC|CD|AC|aC|a$$
$$A \to aAb|ab$$
$$C \to aC|a$$
$$D \to aDa|bDb|aa|bb$$

**Step 4** – Restricting the right sides of productions to single terminals or strings of two or more variables. Thus, replace terminal only right by new variables $B_a \to a$ and $B_b \to b$

$$S \to AACD|ACD|AAC|CD|AC|B_aC|a$$
$$A \to B_aAB_b|B_aB_b$$
$$C \to B_aC|a$$
$$D \to B_aDB_a|B_bDB_b|B_aB_a|B_bB_b$$
$$B_a \to a$$
$$B_b \to b$$

**Step 5** – Replace a combination of two variables on R.H.S. by a single variable unit to get the R.H.S. of the productions in two variables

$S \to AE_1|AE_2|AE_3|CD|AC|B_aC|a$
$A \to B_aE_4|B_aB_b$
$B_a \to a$
$B_b \to b$
$C \to B_aC|a$
$D \to B_aE_5|B_bE_6|B_aB_a|B_bB_b$
$E_1 \to AE_2,\ E_2 \to CD,\ E_3 \to AC,$
$E_4 \to AB_b,\ E_5 \to DB_a,\ E_6 \to DB_b$

***Prob.33. Convert the following CFG to CNF***

$S \to ASB \mid \in$
$A \to aAS \mid a$
$B \to SbS \mid A \mid bb$

***(R.G.P.V., June***

***Sol.*** **Step 1** – Write all the productions separately

$S \to ASB$
$S \to \in$
$A \to aAS$
$A \to a$
$B \to SbS$
$B \to A$
$B \to bb$

The only production $A \to a$ in Chomsky normal form.

**Step 2** – Eliminating $\in$-productions –

$S \to ASB \mid AB$
$A \to aAS \mid aA \mid a$
$B \to SbS \mid bS \mid Sb \mid b \mid A \mid bb$

**Step 3** – Eliminating unit productions –

$S \to ASB \mid AB$
$A \to aAS \mid aA \mid a$
$B \to SbS \mid bS \mid Sb \mid b \mid aAS \mid aA \mid a \mid bb$

**Step 4** – Restricting the right sides of productions to single termi
strings of two or more variables. Thus, replace terminal only righ
variable

$B_a \to a$ and $B_b \to b$
$S \to ASB \mid AB$
$A \to B_aAS \mid B_aA \mid a$
$B \to SB_bS \mid B_bS \mid SB_b \mid b \mid B_aAS \mid B_aA \mid a \mid B_bB_b$



**Step 5** – Replace a combination of two variables on R.H.S. by a single
riable unit to get the R.H.S. of the productions in two variables

$S \to AA_1|AB$
$A \to B_aA_2|B_aA|a$
$A_1 \to SB$
$A_2 \to AS$
$B \to SA_3|B_bS|SB_b|b|B_aA_2|B_aA|a|B_aB_b$
$A_3 \to B_bS$

***Prob.34. Transform the grammar with productions –***

$S \to abAB,\ A \to bAB \mid \lambda,\ B \to BAa \mid A \mid \lambda$

***to Chomsky normal form.*** ***(R.G.P.V., June 2005)***

***Or***

***Transform the grammar with productions –***

$S \to abAB,\ A \to bAB \mid \in,\ B \to BAa \mid A \mid \in$

***to Chomsky normal form.***

***(R.G.P.V., June 2008)***

***Sol.*** **Step 1** – Write all the production separately

$S \to abAB$
$A \to bAB$
$A \to \lambda$
$B \to BAa$
$B \to A$
$B \to \lambda$

**Step 2** – Eliminating $\lambda$-productions

$S \to abAB|abB|abA|ab$
$A \to bAB|bB|bA|b$
$B \to BAa|Ba|Aa|a$

In this grammar, there is no unit production, thus the elimination of unit
roduction is not performed.

**Step 3** – Restricting the right sides of productions to single terminals or
rings of two or more variables. Thus, replace terminal only right by new
ariables $C_a \to a$ and $C_b \to b$.

$S \to C_aC_bAB|C_aC_bB|C_aC_bA|C_aC_b$
$A \to C_bAB|C_bB|C_bA|b$
$B \to BAC_a|BC_a|AC_a|a$
$C_a \to a$
$C_b \to b$

**Step 4** – Replace a combination of two variables on R.H.S. by a single
riable, until the R.H.S. of the productions have only two variables.

$S \to C_aE_1,\ E_1 \to C_bE_2,\ E_2 \to AB$
$S \to C_aE_3,\ E_3 \to C_bB$
$S \to C_aE_4,\ E_4 \to C_bA$
$S \to C_aC_b$
$A \to C_bE_2 \mid C_bB \mid C_bA \mid b$
$B \to BE_5,\ E_5 \to AC_a$
$B \to BC_a \mid AC_a \mid a$
$C_a \to a$
$C_b \to b$

***Prob.35. Convert the following grammar G into Chomsky normal***

$S \to ABAC$
$A \to aA \mid \in$
$B \to bB \mid \in$
$C \to c$

***(R.G.P.V., Dec.***

***Sol.*** **Step 1 – Elimination of Null Productions and Unit Product**

The following grammar is obtained after removing null production.

$S \to ABAC \mid BAC \mid ABC \mid BC \mid AAC \mid AC \mid C$
$A \to aA \mid a$
$B \to bB \mid b$
$C \to c$

Now, unit productions are removed.

$S \to ABAC \mid BAC \mid ABC \mid BC \mid AAC \mid AC \mid c$
$A \to aA \mid a$
$B \to bB \mid b$
$C \to c$

**Step 2 – Elimination of Terminals on R.H.S. –**

$S \to ABAC \mid BAC \mid ABC \mid BC \mid AAC \mid AC \mid c$
$A \to C_aA \mid a$
$C_a \to a$
$B \to B_bB \mid b$
$B_b \to b$
$C \to c$

**Step 3 – Restricting the Number of Variables on R.H.S. –**

$S \to ABAC$
$S \to AC_1$ where $C_1 \to BAC$ gives rise to $C_1 \to BC_2$ where
$S \to BAC$
$S \to BC_2$ where $C_2 \to AC$



$S \to ABC$
$S \to AC_3$ where $C_3 \to BC$
$S \to AAC$
$S \to AC_2$ where $C_2 \to AC$

The grammar is in CNF.

***Prob.36. Convert the following grammar to CNF –***

$S \to bA \mid aB$
$A \to bAA \mid aS \mid a$
$B \to aBB \mid bSbb$

***(R.G.P.V., Dec. 2011)***

***Sol.*** **Step 1** – Write all the productions separately

$S \to bA$
$S \to aB$
$A \to bAA$
$A \to aS$
$A \to a$
$B \to aBB$
$B \to bSbb$

The only production already in CNF are $A \to a$. There are no null and unit productions.

**Step 2** – Choose the productions which are not in CNF, they are

$S \to bA,\ S \to aB,\ A \to bAA,\ A \to aS,\ B \to aBB,\ B \to bSbb$

**Step 3** – Replace terminal only right by new variables $C_a \to a$ and $C_b \to b$

(i) $S \to bA$ is replaced by $S \to C_bA$ such that $C_b \to b$
(ii) $S \to aB$ is replaced by $S \to C_aB$ such that $C_a \to a$
(iii) $A \to bAA$ is replaced by $A \to C_bAA$ such that $C_b \to b$
(iv) $A \to aS$ is replaced by $A \to C_aS$ such that $C_a \to a$
(v) $B \to aBB$ is replaced by $B \to C_aBB$ such that $C_a \to a$
(vi) $B \to bSbb$ is replaced by $B \to C_bSC_bC_b$ such that $C_b \to b$

**Step 4** – Replaced a combination of two variables on R.H.S. by a single variable.

(i) $A \to C_bAA$ is replaced by $A \to C_bD_1$ such that $D_1 \to AA$
(ii) $B \to C_aBB$ is replaced by $B \to C_aD_2$ such that $D_2 \to BB$
(iii) $B \to C_bSC_bC_b$ is replaced by $B \to C_bSD_3$ such that $D_3 \to C_bC_b$

**Step 5** – Again replaced a combination of two variables on R.H.S. by a single variable.

$B \to C_bSD_3$ is replaced by $B \to C_bE_1$ such that $E_1 \to SD_3$

Therefore the grammar $G'$ is

$G' = (\{S, A, B, C_a, C_b, D_1, D_2, D_3, E_1\}, \{a, b\}, P_1, S)$

The production $P_1$ which are in CNF are

$S \to C_bA \mid C_aB$
$A \to C_bD_1 \mid C_aS \mid a$
$B \to C_aD_2 \mid C_bE_1$
$C_a \to a$
$C_b \to b$
$D_1 \to AA$
$D_2 \to BB$
$D_3 \to C_bC_b$
$E_1 \to SD_3$

***Prob.37. Reduce the grammar into CNF and GNF –***

$G = (\{S, A, B\}, \{0, 1\}, \{S \to 1A \mid 0B, A \to 0S \mid 0, B \to 1S \mid$

***(R.G.P.V., Dec***

***Sol.*** Given grammar is

$S \to 1A|0B$
$A \to 0S|0$
$B \to 1S|1$

***(i) Conversion into CNF***

Let $A_1 \to 0$ and $A_2 \to 1$

Then the above grammar in Chomsky normal form will be

$S \to A_2A|A_1B, A_1 \to 0$
$A \to A_1S|0, A_2 \to 1$
$B \to A_2S|1$

***(ii) Conversion into GNF***

The given grammar is already in *GNF*

i.e.

$S \to 1A|0B$
$A \to 0S|0$
$B \to 1S|1$

***Prob.38. Construct a grammar in GNF equivalent to gramma***

$S \to AA|a$
$A \to SS|b$

***(R.G.P.V., Jun***

***Sol.*** **Step 1** – The given grammar is in CNF. $S$ and $A$ are renam and $A_2$, respectively. So the productions are $A_1 \to A_2A_2 \mid a$ and $A_2 \to$



As the given grammar has no null productions and is in CNF we need not carry out step 1. So we proceed to step 2.

**Step 2** – (i) $A_1$-productions are in the required form. They are $A_1 \to A_2A_2 \mid a$.

(ii) $A_2 \to b$ is in the required form. We apply Lemma 1 of GNF and we get, they are $A_2 \to A_1A_1$. The resulting productions are $A_2 \to A_2A_2A_1$, $A_2 \to aA_1$. Thus the $A_2$-productions $A_2 \to A_2A_2A_1$, $A_2 \to aA_1$, $A_2 \to b$.

**Step 3** – $A_2$-productions as we have $A_2 \to A_2A_2A_1$. Let $Z_2$ be the new variable. The resulting productions are

$A_2 \to aA_1, \quad A_2 \to b$
$A_2 \to aA_1Z_2, \quad A_2 \to bZ_2$
$Z_2 \to A_2A_1, \quad Z_2 \to A_2A_1Z_2.$

**Step 4** – (i) $A_2$-productions are $A_2 \to aA_1 \mid b \mid aA_1Z_2 \mid bZ_2$.

(ii) Among the $A_1$-productions we retain $A_1 \to a$ and eliminate $A_1 \to A_2A_2$ using Lemma 1 of GNF. The resulting productions are $A_1 \to aA_1A_2 \mid bA_2, A_1 \to aA_1Z_2A_2 \mid bZ_2A_2$. The set of all (modified) $A_1$-productions are $A_1 \to a|aA_1A_2|bA_2|aA_1Z_2A_2|bZ_2A_2$.

**Step 5** – The $Z_2$-productions to be modified are $Z_2 \to A_2A_1$, $Z_2 \to A_2A_1Z_2$. We apply Lemma 1 of GNF get

$Z_2 \to aA_1A_1 \mid bA_1 \mid aA_1Z_2A_1 \mid bZ_2A_1$
$Z_2 \to aA_1A_1Z_2 \mid bA_1Z_2 \mid aA_1Z_2A_1Z_2 \mid bZ_2A_1Z_2$

Hence, the equivalent grammar is

$G' = (\{A_1, A_2, Z_2\}, \{a, b\}, P_1, A_1)$

where $P_1$ consists of

$A_1 \to a \mid aA_1A_2 \mid bA_2 \mid aA_1Z_2A_2 \mid bZ_2A_2$
$A_2 \to aA_1 \mid b \mid aA_1Z_2 \mid bZ_2$
$Z_2 \to aA_1A_1 \mid bA_1 \mid aA_1Z_2A_1 \mid bZ_2A_1$
$Z_2 \to aA_1A_1Z_2 \mid bA_1Z_2 \mid aA_1Z_2A_1Z_2 \mid bZ_2A_1Z_2$

***Prob.39. Find a Greibach normal form grammar equivalent to the following CFG –***

$S \to AA|0$
$A \to SS|1$

***Or***

***Convert the following grammar into Greibach normal form.***

$S \to AA \mid 0$
$A \to SS \mid 1$

***(R.G.P.V., Dec. 20***

***Or***

***Find the Greibach normal form grammar equivalent to the follow***

***CFG –***

$S \to XX \mid 0$
$X \to SS \mid 1$ **(R.G.P.V., Dec.**

**Or**

***Convert the following CFG into Greibach normal form –***

$S \to AA|b$
$A \to SS|a$ **(R.G.P.V., June**

**Or**

***Convert the following grammar into GNF.***

$S \to AA \mid 0$
$A \to SS \mid 1$ **(R.G.P.V., Nov.**

***Sol.*** Refer to Prob.38.

***Prob.40. Convert the following grammar G into GNF***

$S \to XA \mid BB$
$B \to b \mid SB$
$X \to b$
$A \to a$

**(R.G.P.V., June**

***Sol.*** **Step 1** – The given grammar is in CNF. $S$, $B$, $X$ and $A$ are renamed $A_1$, $A_2$, $A_3$ and $A_4$, respectively. So the productions are $A_1 \to A_3A_4 \mid A$, $A_2 \to b \mid A_1A_2$, $A_3 \to b$, $A_4 \to a$. Now, we proceed to step 2.

**Step 2** – $A_1$ productions are in the required form. They are $A_1 \to A$, $A_2A_2$. The $A_3 \to b$ production is in the required form. The $A_4 \to a$ produc is also in required form. The production $A_2 \to b \mid A_1A_2$ are not in req form. Thus the required productions are as follows –

$A_2 \to b$, $A_2 \to A_3A_4A_2$, $A_2 \to A_2A_2A_2$

Again, substitute $A_3 \to b$, we get

$A_2 \to bA_4A_2$, $A_2 \to A_2A_2A_2$

**Step 3** – The $A_2$ production as we have $A_2 \to A_1A_2A_2A_2$

Let $Z_4$ be the new variable. The resulting productions are –

$A_2 \to b$, $A_2 \to bA_4A_2$
$A_2 \to bZ$, $A_2 \to bA_4A_2Z$
$Z \to A_2A_2$, $Z \to A_2A_2Z$

**Step 4** – $A_2$ productions are –

$A_2 \to b \mid bA_4A_2 \mid bZ \mid bA_4A_2Z$

Among the productions $A_1$, we eliminate $A_1 \to A_3A_4 \mid A_2A_2$. The res productions are $A_1 \to bA_4$, $A_1 \to bA_2$, $A_1 \to bA_4A_2A_2$, $A_1 \to bZA_2$, $bA_4A_2ZA_2$. The set of all (modified) $A_1$ productions are $A_1 \to bA_4 \mid$ $bA_4A_2A_2 \mid bZA_2 \mid bA_4A_2ZA_2$.



**Step 5** – The $Z$ productions to be modified are –

$Z \to A_2A_2$ and $Z \to A_2A_2Z$, we get

$Z \to bA_2 \mid bA_4A_2A_2 \mid bZA_2 \mid bA_4A_2ZA_2$
$Z \to bA_2Z \mid bA_4A_2A_2Z \mid bZA_2Z \mid bA_4A_2ZA_2Z$

Hence, the equivalent grammar is

$G' = (\{A_1, A_2, A_3, A_4\}, (a, b), P_1, A_1)$

where $P_1$ consists of

$A_1 \to bA_4 \mid bA_2 \mid bA_4A_2A_2 \mid bZA_2 \mid bA_4A_2ZA_2$
$A_2 \to b \mid bA_4A_2 \mid bZ \mid bA_4A_2Z$
$A_3 \to b$
$A_4 \to a$
$Z \to bA_2 \mid bA_4A_2A_2 \mid bZA_2 \mid bA_4A_2ZA_2 \mid bA_2Z \mid bA_4A_2A_2Z \mid$
$bZA_2Z \mid bA_4A_2ZA_2Z$

***Prob.41. Convert the following CFG to Greibach Normal Form (GNF) –***

$S \to AB \mid 0$
$A \to BC \mid 1$
$B \to CD \mid 2$
$C \to AD \mid 0$
$D \to 1$ **(R.G.P.V., June 2007, Dec. 2009)**

***Sol.*** **Step-1** – The given grammar is in CNF. $S$, $A$, $B$, $C$ and $D$ are renamed as $A_1$, $A_2$, $A_3$, $A_4$ and $A_5$, respectively. So, the productions are $A_1 \to A_2A_3|0$, $A_2 \to A_3A_4 \mid 1$, $A_3 \to A_4A_5|2$, $A_4 \to A_2A_5|0$ and $A_5 \to 1$. Now we proceed to next step.

**Step-2 –**

(i) $A_1$-productions are in the required form. They are $A_1 \to A_2A_3 \mid 0$.

(ii) The $A_2 \to A_3A_4 \mid 1$ are in the required form.

(iii) $A_3 \to A_4A_5 \mid 2$ are in the required form.

(iv) $A_5 \to 1$ is also in required form.

The productions $A_4 \to A_2A_5 \mid 0$ are not in required form. Thus the required productions are as follows –

$A_4 \to A_3A_4A_5$, $A_4 \to 1A_5$, $A_4 \to 0$

Again, eliminating $A_4 \to A_3A_4A_5$, we get,

$A_4 \to A_4A_5A_4A_5$, $A_4 \to 2A_4A_5$

**Step-3** – The $A_4$-production as we have $A_4 \to A_4A_5A_4A_5$. Let $Z_4$ be the new variable. The resulting productions are –

$A_4 \to 1A_5$ $A_4 \to 0$
$A_4 \to 2A_4A_5$ $A_4 \to 0Z_4$

$A_4 \to 1A_5\ Z_4 \quad A_4 \to 2A_4A_5Z_4$

$Z_4 \to A_5A_4A_5 \quad Z_4 \to A_5A_4A_5Z_4$

**Step-4** – (i) $A_4$-productions are as –

$A_4 \to 0 \mid 1A_5 \mid 2\ A_4A_5 \mid 0Z_4 \mid 1\ A_5Z_4 \mid 2A_4A_5Z_4$

(ii) Among $A_3$-productions, we retain

$A_3 \to 2$ and eliminate $A_3 \to A_4A_5$, we get

$A_3 \to 0A_5 \mid 1A_5\ A_5 \mid 2A_4\ A_5\ A_5 \mid 0Z_4\ A_5 \mid 1\ A_5\ Z_4\ A_5 \mid 2A_4A_5$

The modified $A_3$-productions are

$A_3 \to 2 \mid 0A_5 \mid 1\ A_5\ A_5 \mid 2\ A_4\ A_5\ A_5 \mid 0\ Z_4\ A_5 \mid 1\ A_5\ Z_4\ A_5 \mid 2A_4A_5Z_4$

(iii) Similarly, among $A_2$-productions,

$A_2 \to 1 \mid 2A_4 \mid 0A_5\ A_4 \mid 1A_5\ A_5\ A_4 \mid 2A_4A_5A_5A_4 \mid 0Z_4A_5A_4 \mid 1A_5Z_4A_5A_4 \mid 2A_4A_5Z_4A_5A_4$

(iv) Among $A_1$-productions, we have

$A_1 \to 0 \mid 1A_3 \mid 2A_4A_3 \mid 0A_5A_4A_3 \mid 1\ A_5A_5A_4A_3 \mid 2\ A_4A_5A_5A_4A_3 \mid 0Z_4A_5A_4A_3 \mid 1A_5Z_4A_5A_4A_3 \mid 2A_4A_5Z_4A_5A_4A_3$

(v) Among $A_5$-productions,

$A_5 \to 1$

**Step-5** – The $Z_4$-production to be modified are, $Z_4 \to A_5A_4A_5 \mid A_5A_4A_5Z_4$, we get,

$Z_4 \to 1A_4A_5 \mid 1A_4A_5Z_4$

The required GNF is given by equations (i) to (vi).

***Prob.42. Find a Greibach normal form equivalent to the following CFG –***

$S \to AB|a$

$A \to BS|b$

$B \to SA|c$

***(R.G.P.V., Dec. 20***

***Sol.*** Since the given grammar is in CNF we can omit step 1 and proceed to step 2 after renaming $S, A, B$ as $A_1, A_2, A_3$, respectively. The productions are $A_1 \to A_2A_3|a$, $A_2 \to A_3A_1|b$, $A_3 \to A_1A_2|a$.

**Step 2** – (i) The $A_1$-productions $A_1 \to A_2A_3|a$ are in the required form.

(ii) The $A_2$-productions $A_2 \to A_3A_1|b$ are in the required form.

(iii) $A_3 \to c$ is in the required form.



Applying lemma to $A_3 \to A_1A_2$. The resulting productions are $A_3 \to A_2A_3A_2|aA_2$. Applying the lemma once again to $A_3 \to A_2A_3A_2$, we get $A_3 \to A_2A_3A_2|aA_2|bA_3A_2|bA_3A_1A_3A_2$.

**Step 3** – The $A_3$-productions are $A_3 \to c|bA_3A_2|aA_2$ and $A_3 \to A_3A_1A_3A_2$. As we have $A_3 \to A_3A_1A_3A_2$, we have to apply lemma to $A_3$-productions. Let $Z_3$ be the new variable. The resulting productions are

$A_3 \to c|bA_3A_2|aA_2,\ A_3 \to cZ_3|bA_3A_2Z_3|aA_2Z_3$

$Z_3 \to A_1A_3A_2,\ Z_3 \to A_1A_3A_2Z_3$

**Step 4** – (i) The $A_3$-productions are

$A_3 \to c|bA_3A_2|cZ_3|bA_3A_2Z_3|aA_2|aA_2Z_3$ ...(i)

(ii) Among $A_2$-productions, we retain $A_2 \to b$ and eliminate $A_2 \to A_3A_1$ using lemma. The resulting productions are

$A_2 \to cA_1|bA_3A_2A_1|cZ_3A_1|bA_3A_2Z_3A_1|aA_2A_1|aA_2Z_3A_1$

The modified $A_2$-productions are

$A_2 \to b|cA_1|bA_3A_2A_1|cZ_3A_1|bA_3A_2Z_3A_1|aA_2A_1|aA_2Z_3A_1$ ...(ii)

(iii) We apply lemma to $A_1 \to A_2A_3$ to get

$A_1 \to bA_3|cA_1A_3|bA_3A_2A_1A_3|cZ_3A_1A_3|\ bA_3A_2Z_3A_1A_3 \mid aA_2A_1A_3|aA_2Z_3A_1A_3$ ...(iii)

**Step 5** – The $Z_3$-productions to be modified are

$Z_3 \to bA_3A_3A_2|bA_3A_3A_2Z_3$

$Z_3 \to cA_1A_3A_3A_2|cA_1A_3A_3A_2Z_3$

$Z_3 \to bA_3A_2A_1A_3A_3A_2|bA_3A_2A_1A_3A_3A_2Z_3$

$Z_3 \to cZ_3A_1A_3A_3A_2|cZ_3A_1A_3A_3A_2Z_3$

$Z_3 \to bA_3A_2Z_3A_1A_3A_3A_2|bA_3A_2Z_3A_1A_3A_3A_2Z_3$

$Z_3 \to aA_2A_1A_3A_3A_2|aA_2A_1A_3A_3A_2Z_3$

$Z_3 \to aA_2Z_3A_1A_3A_3A_2|aA_2Z_3A_1A_3A_3A_2Z_3$ ...(iv)

The required grammar in GNF is given by equations (i) to (iv).

***Prob.43. Convert the following grammar into GNF.***

$A_1 \to A_2A_3$

$A_2 \to A_3A_1 \mid b$

$A_3 \to A_1A_2 \mid a$

***(R.G.P.V., Dec. 2013)***

***Or***

***Convert the following CFG to GNF –***

$A \to BC$

$B \to CA \mid b$

$C \to AB \mid a$

***(R.G.P.V., Dec. 2014)***

**Or**

***Convert the grammar $S \to AB$, $A \to BS \mid b$, $B \to SA \mid a$ into Greibach normal form.*** ***(R.G.P.V., June 201***

***Sol.*** The grammar is

$$G = (\{A_1, A_2, A_3\}, \{a, b\}, P, A_1)$$

where P consists of the following –

$A_1 \to A_2A_3$
$A_2 \to A_3A_1 \mid b$
$A_3 \to A_1A_2 \mid a$

The steps to convert the grammar into GNF are as follows –

**Step-1** – Since the right-hand side of the productions for $A_1$ and $A_2$ begin with terminals or higher-numbered variables, we start with the production $A_3 \to A_1A_2$ and substitute the string $A_2A_3$ for $A_1$.

The resulting set of productions is

$A_1 \to A_2A_3$
$A_2 \to A_3A_1 \mid b$
$A_3 \to A_2A_3A_2 \mid a$

Since the right side of the production $A_3 \to A_2A_3A_2$ begins with a lower-numbered variable, we substitute for the first occurrence of $A_2$ both $A_3A_1$ and $b$. Thus, the new set is

$A_1 \to A_2A_3$
$A_2 \to A_3A_1 \mid b$
$A_3 \to A_3A_1A_3A_2 \mid bA_3A_2 \mid a$

Now, apply lemma to the productions

$A_3 \to A_3A_1A_3A_2 \mid bA_3A_2 \mid a$

Symbol $B_3$ is introduced, and the production $A_3 \to A_3A_1A_3A_2$ is replaced by $A_3 \to bA_3A_2B_3$, $A_3 \to aB_3$, $B_3 \to A_1A_3A_2$, and $B_3 \to A_1A_3A_2B_3$. The resulting set is

$A_1 \to A_2A_3$
$A_2 \to A_3A_1 \mid b$
$A_3 \to bA_3A_2B_3 \mid aB_3 \mid bA_3A_2 \mid a$
$B_3 \to A_1A_3A_2 \mid A_1A_3A_2B_3$

**Step-2** – Now all the productions with $A_3$ on the left have right-hand sides that start with terminals. These are used to replace $A_3$ in the production $A_2 \to A_3A_1$ and then the productions with $A_2$ on the left are used to replace $A_2$ in the production $A_1 \to A_2A_3$. The result is as follows –

| | |
|---|---|
| $A_3 \to bA_3A_2B_3$ | $A_3 \to bA_3A_2$ |
| $A_3 \to aB_3$ | $A_3 \to a$ |



| | |
|---|---|
| $A_2 \to bA_3A_2B_3A_1$ | $A_2 \to bA_3A_2A_1$ |
| $A_2 \to aB_3A_1$ | $A_2 \to aA_1$ |
| $A_2 \to b$ | |
| $A_1 \to bA_3A_2B_3A_1A_3$ | $A_1 \to bA_3A_2A_1A_3$ |
| $A_1 \to aB_3A_1A_3$ | $A_1 \to aA_1A_3$ |
| $A_1 \to bA_3$ | |
| $B_3 \to A_1A_3A_2$ | $B_3 \to A_1A_3A_2B_3$ |

**Step-3** – The two $B_3$ productions are converted to proper form resulting in 10 more productions. That is, the productions

$B_3 \to A_1A_3A_2$ and $B_3 \to A_1A_3A_2B_3$

are changed by substituting the right side of each of the five productions with $A_1$ on the left for the first occurrences of $A_1$. Thus, $B_3 \to A_1A_3A_2$ becomes

$B_3 \to bA_3A_2B_3A_1A_3A_3A_2$, $B_3 \to aB_3A_1A_3A_3A_2$
$B_3 \to bA_3A_3A_2$, $B_3 \to bA_3A_2A_1A_3A_3A_2$, $B_3 \to aA_1A_3A_3A_2$

The other production for $B_3$ is replaced similarly. The final set of productions is

| | |
|---|---|
| $A_3 \to bA_3A_2B_3$ | $A_3 \to bA_3A_2$ |
| $A_3 \to aB_3$ | $A_3 \to a$ |
| $A_2 \to bA_3A_2B_3A_1$ | $A_2 \to bA_3A_2A_1$ |
| $A_2 \to aB_3A_1$ | $A_2 \to aA_1$ |
| $A_2 \to b$ | |
| $A_1 \to bA_3A_2B_3A_1A_3$ | $A_1 \to bA_3A_2A_1A_3$ |
| $A_1 \to aB_3A_1A_3$ | $A_1 \to aA_1A_3$ |
| $A_1 \to bA_3$ | |
| $B_3 \to bA_3A_2B_3A_1A_3A_3A_2B_3$ | $B_3 \to bA_3A_2B_3A_1A_3A_3A_2$ |
| $B_3 \to aB_3A_1A_3A_3A_2B_3$ | $B_3 \to aB_3A_1A_3A_3A_2$ |
| $B_3 \to bA_3A_3A_2B_3$ | $B_3 \to bA_3A_3A_2$ |
| $B_3 \to bA_3A_2A_1A_3A_3A_2B_3$ | $B_3 \to bA_3A_2A_1A_3A_3A_2$ |
| $B_3 \to aA_1A_3A_3A_2B_3$ | $B_3 \to aA_1A_3A_3A_2$ |

***Prob.44. Reduce the following grammar to Greibach normal form–***

$S \to AO$, $A \to OB$, $B \to AO$, $B \to 1$ ***(R.G.P.V., Dec. 2009)***

***Sol.***

$S \to AO$
$A \to OB$
$B \to AO \mid 1$

Here, $S \to AO$ and $B \to AO$ is not in *GNF*

So, put $A \to OB$ in $S \to AO$ & $B \to AO$ we get,

$S \to OBO$

$A \to OB$

$B \to OBO|1$

Now, let $A_1 \to O$

So the above grammar will be

$S \to OBA_1$

$A \to OB$

$B \to OBA_1|1$

which is in Greibach normal form.

## PUMPING LEMMA FOR CFLs, DECISION ALGORITHMS FO CFGs, DESIGNING CFGs, CLOSURE PROPERTIES OF CFLs

***Q.16. What is the purpose of pumping lemma for CFG ?***

***Ans.*** The pumping lemma for context-free languages gives a method generating infinite number of strings from a given sufficiently long string context-free language $L$. It is used to prove that certain languages are context-free.

***Q.17. How do we use pumping lemma to show that a language L is a context-free language ?***

***Ans.*** We use pumping lemma to show that a language does not belong the family of context-free languages. Its application is typical of pump lemmas in general and they are used negatively to show that a given langu does not belong to some family.

We assume that $L$ is a context-free language. By applying pumping le we get a contradiction.

The procedure can be carried out by using the following steps –

***(i) Step 1*** – Assume $L$ is context-free. Let $n$ be the natural num obtained by using the pumping lemma.

***(ii) Step 2*** – Choose $u \in L$ so that $|u| \geq n$. Write $u = vwxyz$ using pumping lemma.

***(iii) Step 3*** – Find a suitable $k$ so that $vw^kxy^kz \notin L$. This contradiction, and so $L$ is not context-free.

***Q.18. State and prove pumping lemma for CFG. (R.G.P.V., June***

***Or***

***Explain pumping lemma for CFL. (R.G.P.V., June***



***Explain the pumping Lemma for context free languages.***
***(R.G.P.V., Dec. 2016)***

***Ans.*** Let $L$ be an infinite context-free language. Then there exists some positive integer $m$ such that any $u \in L$ with $|u| \geq m$ can be decomposed as –

$$u = vwxyz, \quad \text{...(i)}$$

$$|wxy| \leq m \quad \text{...(ii)}$$

with $|wy| \geq 1$ ...(iii)

and $vw^i xy^i z \in L$ ...(iv)

such that

for all $i = 0, 1, 2, \ldots$. This is known as the pumping lemma for context-free languages.

**Proof** – Consider the language $L - \{\Lambda\}$, and let we have for it a grammar $G$ without unit-productions or null-productions. Since the length of the string on the right-side of any production is bounded, say by $k$, the length of the derivation of any $u \in L$ must be atleast $|u|/k$. Therefore, since $L$ is infinite, there exist arbitrarily long derivations and corresponding derivation trees of arbitrary height.

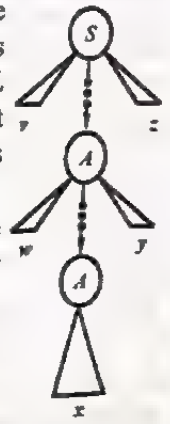



Fig. 3.19

Now consider such a high derivation tree and some sufficiently long path from the root to a leaf. Since the number of variables in $G$ is finite, there must be some variable that repeats on this path. This is shown schematically in fig. 3.19.

Corresponding to the derivation tree in fig. 3.19, we have the derivation

$$S \overset{*}{\Rightarrow} vAz \overset{*}{\Rightarrow} vwAyz \overset{*}{\Rightarrow} vwxyz$$

where $v$, $w$, $x$, $y$ and $z$ are all strings of terminals. From the above we can see that $A \overset{*}{\Rightarrow} wAy$ and $A \overset{*}{\Rightarrow} x$, so all the strings $vw^ixy^iz$, $i = 0, 1, 2, \ldots$ can be generated by the grammar and are therefore in $L$. Furthermore, in the derivations $A \overset{*}{\Rightarrow} wAy$ and $A \overset{*}{\Rightarrow} x$, we can assume that no variable repeats (otherwise, we just use the repeating variable as $A$). Therefore the lengths of the strings $w$, $x$ and $y$ depend only on the productions of the grammar and can be bounded independently of $u$ so that (ii) is proved. Finally, since there are no unit-productions and no null-productions, $w$ and $y$ cannot be empty strings and this proves (iii).

This completes the argument that (i) to (iv) hold.

***Q.19. Write down decision algorithms for context-free languages.***

***Or***

***Write short note on decision algorithms. (R.G.P.V., June 2003, 2004)***

***Ans.*** Some decision algorithms for context-free languages and regular sets are given below –

(i) Algorithm for deciding whether a context-free language $L$ is empty

**Proof** – For simplicity, assume that $\Lambda \notin L(G)$. Slight changes have to be made in the argument if this is not so. Use the algorithm for removing useless symbols and productions. If $S$ is found to be useless, then $L(G)$ is empty; if not, then $L(G)$ contains at least one element.

(ii) Algorithm for deciding whether a context-free language is infinite

**Proof** – We assume that $G$ contains no $\Lambda$-productions, no unit productions and no useless symbols. Suppose the grammar has a repeating variable in the sense that there exists some $A \in V$ for which there is a derivation

$$A \overset{*}{\Rightarrow} xAy.$$

Since $G$ is supposed to have no $\Lambda$-productions and no unit productions, $x$ and $y$ cannot be simultaneously empty. Since $A$ is neither nullable nor a useless symbol, we have –

$$S \overset{*}{\Rightarrow} uAv \overset{*}{\Rightarrow} w \text{ and } A \overset{*}{\Rightarrow} z$$

where, $u$, $v$ and $z$ are in $T^*$. But then

$$S \overset{*}{\Rightarrow} uAv \overset{*}{\Rightarrow} ux^nAy^nv \overset{*}{\Rightarrow} ux^nzy^nv$$

is possible for all $n$, such that $L(G)$ is infinite.

If no variable can ever repeat, so the length of any derivation is bounded by $|V|$. In such case, $L(G)$ is finite.

Thus, to get an algorithm for finding whether $L(G)$ is finite, we need only to determine whether the grammar has any repeating variables. This can be done by simply drawing a dependency graph for the variables such that there is an edge $(A, B)$ whenever there is a corresponding production –

$$A \to xBy$$

Then any variable at the base of a cycle is a repeating one. Consequently, the grammar has a repeating variable, iff the dependency graph has a cycle.

As we now have an algorithm for determining whether a grammar has a repeating variable, we have an algorithm for deciding whether or not $L(G)$ is infinite.

(iii) Algorithm for deciding whether a context-free language $L$ is finite –

**Proof** – Construct a nonredundant context-free grammar $G$ in CNF generating $L-\{\Lambda\}$. We draw a directed graph whose vertices are variables of $G$. If $A \to BC$ is a production, there are directed edges from $A$ to $B$ and $A$ to $C$. $L$ is finite if and only if the directed graph has no cycles.

(iv) Algorithm for deciding whether a regular language $L$ is empty

**Proof** – Construct a deterministic finite automaton $M$ accepting $L$. We find / construct the set of all states reachable from the initial state $q_0$. We find



states which are reachable from $q_0$ by applying a single input symbol. These states are arranged as a row under columns corresponding to every input symbol. The construction is repeated for every state appearing in an earlier row. The construction terminates in a finite number of steps. If a final state appears in this tabular column, then $L$ is nonempty. (Actually, we can terminate the construction as soon as some final state is obtained in the tabular column.) Otherwise, $L$ is empty.

(v) Algorithm for deciding whether a regular language $L$ is infinite–

Construct a deterministic finite automaton $M$ accepting $L$. $L$ is infinte if and only if $M$ has a cycle.

***Q.20. Briefly explain the designing of context-free grammar.***

***Ans.*** Convensions regarding designing of grammar –

(i) The capital letters denote variables like

$A$, $B$, $C$, $D$, $E$ and $S$.

Here $S$ is the start symbol, if not then stated.

(ii) Digits, bold face strings and lower-case letters $a$, $b$, $c$, $d$, and $e$ are terminals.

(iii) $X$, $Y$, $Z$ capital letters denote symbols either terminals or variables.

(iv) $\alpha$, $\beta$, and $\gamma$ (lower-case greek letter) are strings of terminals and variables.

(v) $u$, $v$, $w$, $x$, $y$ and $z$ denote strings of terminals.

These conventions are used to recognise elements of grammar. We can easily differentiate among variables, terminals and start symbol only by examine the productions.

Now we can simply write the productions as

$A \to \alpha_1$, $A \to \alpha_2$, $A \to \alpha_3$ ..... $A \to \alpha_n$

These are the productions for variable $A$ of some grammar, we can write the above notation as

$A \to \alpha_1|\alpha_2|\alpha_3|.....\alpha_n$

Here vertical line (|) is read as 'or'

For example – The grammar for

$L = \{\omega c \omega^R | \omega \in \{a, b\}^*\}$ will be

$S \to aSa$

$S \to bSb$

$S \to c$

Combinely

$S \to aSa|bSb|c$

***Q.21. Write short note on properties of context-free grammar.*** ***(R.G.P.V., Dec. 2005, June 2***

***Or***

***Define the closure properties of CFL's.*** ***(R.G.P.V., Dec. 2***

***Or***

***Write the closure properties of CFL's.*** ***(R.G.P.V., Dec. 2***

***Ans.*** There are some operations that preserve context-free langua The operations are useful not only in constructing or proving that cer languages are context-free, but also in proving certain languages not to context-free. A given language $L$ can be shown not to be context-free constructing from $L$ a language that is not context-free using only operati preserving CFL's. CFL's have the following closure properties –

(i) The CFL's are closed under union, concatenation and Kleen clos

(ii) The CFL's are closed under substitution.

(iii) The CFL's are closed under homomorphism and inve homomorphism.

(iv) The CFL's are not closed under intersection and complementa

(v) Intersection of a CFL and a regular set is a CFL.

***Q.22. Prove that "the CFL's are closed under substitution".***

***Ans.*** **Proof** – Let $L$ be a CFL, $L \subseteq T^*$, and for each $a$ in $T$ let $L_a$ be a CF Let $L$ be $L(G)$ and for each $a$ in $T$ let $L_a$ be $L(G_a)$. Without loss of general assume that the variables of $G$ and the $G_a$'s are disjoint. Construct a gram $G'$ as follows. The variables of $G'$ are all the variables of $G$ and the $G_a$'s a the terminals of $G'$ are the terminals of the $G_a$'s. The start symbol of $G'$ is start symbol of $G$. The productions of $G'$ are all the productions of the together with those productions formed by taking a production $A \to \alpha$ o and substituting $S_a$, the start symbol of $G_a$, for each instance of an $a$ appearing in $\alpha$.

One should see that since $\{a, b\}$, $\{ab\}$ and $a^*$ are CFL's, the closure CFL's under substitution implies closure under union, concatenation, and The union of $L_a$ and $L_b$ is simply the substitution of $L_a$ and $L_b$ into $\{a, b\}$ similarly $L_a L_b$ and $L_a^*$ are the substitutions into $\{ab\}$ and $a^*$, respectiv

Since a homomorphism is a special type of substitution, we state "the CFL's are closed under homomorphism".

***Q.23. Prove that the family of context-free languages is not clo under intersection and complementations.***

***Or***

***Prove that the CFL's are not closed under intersection.*** ***(R.G.P.V., Dec. 2003, June 2***



***Ans.*** **Proof** – Consider the two languages –

$$L_1 = \{a^n b^n c^m | n \geq 0, m \geq 0\}$$

and

$$L_2 = \{a^n b^m c^m | n \geq 0, m \geq 0\}.$$

There are several ways to prove that $L_1$ and $L_2$ are context-free. For instance, a grammar for $L_1$ is

$$S \to S_1 S_2,$$
$$S_1 \to aS_1 b | \Lambda,$$
$$S_2 \to cS_2 | \Lambda.$$

Alternatively, we note that $L_1$ is the concatenation of two context-free languages, so it is context-free. But

$$L_1 \cap L_2 = \{a^n b^n c^n | n \geq 0\},$$

which is not context-free. Thus, the family of context-free languages is not closed under intersection.

The second part of the question follows the set identity

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}.$$

If the family of context-free languages were closed under complementation, then the R.H.S. of the above expression would be a context-free language for any context-free $L_1$ and $L_2$. But this contradicts what we have just proved that the intersection of two context-free languages is not necessarily context-free. Consequently, the family of context-free languages is not closed under complementation.

***Q.24. The intersection of two context-free languages may or may not be context-free.*** ***(R.G.P.V., Dec. 2015)***

***Ans.*** Given property has two proofs –

(i) We know that all regular language are context free. The intersection of two regular language is also a regular language. Therefore, if $L_1$ and $L_2$ are regular language and context free, then $L_1 \cap L_2$ is both regular and context free.

(ii) Let $L_1 = \{a^n b^n a^m$, where $n, m = 1, 2, 3 \ldots$ but $n$ is not same as $m\}$
$= \{aba, abaa, aabba, \ldots\}$

The CFG for the language $L_1$ is given as

$$S \to XA$$
$$X \to aXb \mid ab$$
$$A \to aA \mid a$$

Here, we have seen that this language is the product of CFL $\{a^n b^n\}$ and the regular $aa^*$. Let $L_2 = \{a^n b^m a^m$, where $n, m = 1, 2, 3, \ldots$, $n$ is not same as $m\}$.
$= \{aba, aaba, abbaa, \ldots\}$

The CFG for language $L_2$ is given as

$$S \to AX$$
$$X \to bXa \mid ba$$
$$a \to aA \mid a$$

Here, $L_2$ is the product of regular language $aa^*$ and CFL $\{b^n a^n\}$.

Both language $L_1$ and $L_2$ are context free but there intersection is ang... $L_3 = L_1 \cap L_2 = \{a^n b^n a^n \text{ for } n = 1, 2, 3, .....\}$.

This language is not context free. Therefore, intersection of two CF... not context free.

***Q.25. Prove that "the family of context-free languages is closed un... union, concatenation, and Kleen closure".***

***Or***

***Prove that CFLs are closed under Kleen closure. (R.G.P.V., Dec. 20...***

***Ans.* Proof** – Let $L_1$ and $L_2$ be two context-free languages generated by... the context-free grammars $G_1 = (V_1, T_1, P_1, S_1)$ and $G_2 = (V_2, T_2, P_2, S_2)$... respectively. Without loss of generality, we can assume that the sets $V_1$ a... $V_2$ are disjoint.

Now consider the language $L(G_3)$, generated by the grammar –

$$G_3 = (V_1 \cup V_2 \cup \{S_3\}, T_1 \cup T_2, S_3, P_3),$$

where $S_3$ is a variable not in $V_1 \cup V_2$. The productions of $G_3$ are all ... productions of $G_1$ and $G_2$, together with an alternative starting produc... that allows us to use one or the other grammars. More precisely,

$$P_3 = P_1 \cup P_2 \cup \{S_3 \to S_1 \mid S_2\}.$$

Obviously, $G_3$ is a context-free grammar, so that $L(G_3)$ is a context-fr... language. But it is easy to observe that

$$L(G_3) = L_1 \cup L_2$$

Suppose for instance that $u \in L_1$. Then

$$S_3 \Rightarrow S_1 \overset{*}{\Rightarrow} u$$

is a possible derivation in grammar $G_3$. A similar argument can be made ... $u \in L_2$. Also, if $u \in L(G_3)$ then either

$$S_3 \Rightarrow S_1$$

or,

$$S_3 \Rightarrow S_2$$

should be the first step of the derivation. Suppose (ii) is used. Since sentential for... derived from $S_1$ have variables in $V_1$, and $V_1$ and $V_2$ are disjoint, the derivation

$$S_1 \overset{*}{\Rightarrow} u$$

can involve productions in $P_1$ only. Hence $u$ must be in $L_1$. Alternatively, if (ii)... used first, then $u$ must be in $L_2$ and it follows that $L(G_3)$ is the union of $L_1$ and...



Now, consider $G_4 = (V_1 \cup V_2 \cup \{S_4\}, T_1 \cup T_2, S_4, P_4)$.

Again $S_4$ is a new variable and

$$P_4 = P_1 \cup P_2 \cup \{S_4 \to S_1 S_2\}.$$

$$L(G_4) = L(G_1)L(G_2)$$

Then,

follows easily.

Finally, consider $L(G_5)$ with

$$G_5 = (V_1 \cup \{S_5\}, T_1, S_5, P_5),$$

where $S_5$ is a new variable and

$$P_5 = P_1 \cup \{S_5 \to S_1 S_5 | \Lambda\}.$$

$$L(G_5) = L(G_1)^*.$$

Then,

Thus we have proved that the family of context-free languages is closed under union, concatenation, and Kleen and star-closure.

***Q.26. Explain closure properties of CFL's*** ***(R.G.P.V., Dec. 2013)***

***Or***

***Explain three closure properties of CFL.***

***(R.G.P.V., June 2011, Dec. 2011)***

***Ans.*** Refer to Q.21, Q.22, Q.23 and Q.25.

***Q.27. If $L_1$ be a context-free language and $L_2$ be a regular language, then prove that $L_1 \cap L_2$ is context-free.***

***Ans.* Proof** – Let $M = (Q_M, \Sigma, \Gamma, \delta_M, q_0, z_0, F_M)$ be an NPDA which accepts $L_1$ and $A = (Q_A, \Sigma, \delta_A, p_0, F_A)$ be a DFA that accepts $L_2$. We construct a pushdown automaton $M' = (Q', \Sigma, \Gamma, \delta', q_0', z_0, F')$ which simulates the parallel action of $M$ and $A$. Whenever a symbol is read from the input string, $M'$ simultaneously executes the moves of $M$ and $A$. To do this we consider –

$$Q' = Q_M \times Q_A,$$
$$q_0' = (q_0, p_0),$$
$$F' = F_M \times F_A,$$

and define $\delta'$ such that

$$((q_k, p_l), x) \in \delta'((q_i, p_j), a, b),$$

if and only if

$$(q_k, x) \in \delta_M(q_i, a, b), \text{ and } \delta_A(p_j, a) = p_l.$$

In this we also require that if $a = \Lambda$, then $p_j = p_l$. In other words, the states of $M'$ are labelled with pairs $(q_i, p_j)$, representing the respective states in which $M$ and $A$ can be after reading a certain input string. It is a straight forward induction argument to prove that

$$(q_0, p_0), w, z_0 \overset{*}{\vdash}_{M'} ((q_r, p_s), x),$$

with $q_r \in F_M$ and $p_s \in F_A$ if and only if

$$(q_0, w, z_0) \vdash_M^* (q_r, x), \text{ and } \delta^*(p_0, w) = p_s.$$

Therefore, a string is accepted by $M'$ if and only if it is accepted by $M$ and $A$, i.e., if it is in $L(M) \cap L(A) = L_1 \cap L_2$.

This property is known as closure under intersection. Because of this result of the theorem, we state that the family of context-free languages is closed under regular intersection. This closure property is sometimes useful for simplifying arguments in connection with specific languages.

## NUMERICAL PROBLEMS

***Prob.45. Show that the language $\{a^{n^2} \mid n \geq 1\}$ is not context free.*** ***(R.G.P.V., Dec. 2014, June 20***

***Sol.*** Let us assume that

$$S = a^{n^2}$$

$S = uvwxy$, where $1 \leq |vx| \leq n$ which is true.

Since $|vwx| \leq n$ (by pumping lemm

Let $|vx| = m$, $m \leq n$

By pumping lemma, $uv^2wx^2y$ is in $L$.

Since $|uv^2wx^2y| > n^2$

$|uv^2wx^2y| = k^2$

where $k \geq n + 1$.

But $|uv^2wx^2y| = n^2 + m < n^2 + 2n + 1$

Therefore, $|uv^2wx^2y|$ lies between $n^2$ and $(n + 1)^2$

Hence, $uv^2wx^2y \notin L$, which is a contradiction.

Therefore $\{a^{n^2} \mid n \geq 1\}$ is not context free.

***Prob.46. State and prove pumping lemma for CFL's. Also prove that $\{a^n b^m a^n b^m : n, m \geq 1\}$ is not context-free.*** ***(R.G.P.V., Dec. 20***

***Sol.*** **Pumping Lemma for CFL** – Refer to Q.18.

**Problem** – $L = \{a^n b^m a^n b^m : n, m \geq 1\}$

Suppose $L$ is a CFL and let $i$ be the integer. Let $u = a^i b^i a^i b^i$. Suppose $v$, $w$, $x$, $y$ and $z$ are any strings satisfying the conditions for pumping lemma. We must derive a contradiction from these facts, without making any other assumptions about the five strings.

$wxy$ can overlap at most two of the four contiguous groups of symbols. We consider several cases.

First, suppose $w$ or $y$ contains at least one $a$ from the first group of $a$'s. Since $|wxy| \leq i$, neither $w$ nor $y$ can contain any symbols from the second



of $u$. Considering $m = 0$, omitting $w$ and $y$ causes at least one initial $a$ to be omitted and does not affect the second half. In other words

$$vw^0xy^0z = a^jb^ka^ib^i$$

where $j < i$ and $k \leq i$. If this string were of the form $ss$ (i.e., if it were in $L$), then unless $j = 0$, $s$ would have to start with $a$ (from the first $s$) and end with $b$ (from the second). This obviously is impossible. If $j = 0$, however, then the string is $b^ia^ib^i$, and this is not of the form $ss$ for any $s$. Since $vw^0xy^0z \notin L$, we have a contradiction in this case.

Next, suppose that $wxy$ contains no $a$'s from the first group but that $w$ or $y$ contains at least one $b$ from the first group of $b$'s. Again, we consider $m = 0$; this time we can say that

$$vw^0xy^0z = a^ib^ja^kb^j$$

for some $j < i$ and some $k \leq i$. As before, if this string were $ss$, $s$ would have to begin with $a$ and end with $b$, and this is impossible. We can conclude from this that $L$ is not a context-free language.

***Prob.47. Show that the language $L = \{a^ib^jc^k / i < j$ and $j < k\}$ is not context free language.*** ***(R.G.P.V., June 2011)***

***Sol.*** Suppose n be constant of pumping lemma.

Choose $z = a^nb^{n+1}c^{n+2}$. This ensures that $z$ is in $L$ and $|z| \geq n$.

If we write $z = uvwxy$, then the possible choices of $vx$ satisfying the conditions –

$1 \leq |vx| \leq n$ and $|vwx| \leq n$ are –

(i) $vx = a^p$, where $1 \leq p \leq n$, i.e., $vx$ contains only $a$'s.

For this choice of $vx$, $uv^2wx^2y$ will be $a^{n+p}b^{n+1}c^{n+2}$ and since $1 \leq p \leq n$, $uv^0wx^0y$ contains more or equal number of $a$'s as comparison to $b$'s. Thus, $uv^2wx^2y$ cannot belong to $L$. Hence contradiction to pumping lemma.

(ii) $vx = b^p$, where $1 \leq p \leq n$, i.e., $vx$ contains only $b$'s.

For this choice of $vx$, $uv^0wx^0y$ will be $a^nb^{n+1-p}c^{n+2}$ and since $1 \leq p \leq n$ $uv^0wx^0y$ contains less or equal number of $b$'s as comparison to $a$'s. Thus, $uv^0wx^0y$ cannot belong to $L$. Hence, contradiction to pumping lemma.

(iii) $vx = c^p$, where $1 \leq p \leq n$, i.e., $vx$ contains only $c$'s.

For this choice of $vx$, $uv^0wx^0y$ will be $a^nb^{n+1}c^{n+2-p}$ and since $1 \leq p \leq n$, $uv^0wx^0y$ contains less or equal number of $c$'s as comparison to $b$'s. Hence, $uv^0wx^0y$ cannot belong to $L$. Hence, contradiction to pumping lemma.

(iv) $vx = a^pb^q$, where $1 \leq p, q \leq n$, i.e., $vx$ contains both $a$'s and $b$'s.

For this choice of $vx$ $uv^2wx^2y$ will be $a^{n+p}b^{n+1+q}c^{n+2}$ and since $1 \leq p, q \leq n$, $uv^2wx^2y$ contains more or equal number of $b$'s as comparison to $a$'s. Hence, $uv^2wx^2y$ cannot belong to $L$. Thus, contradiction to pumping lemma.

(v) $vx = b^p c^q$, where $1 \le p, q \le n$, i.e., $vx$ contains both $b$'s

For this choice of $vx$, $uv^0wx^0y$ will be $a^n b^{n+1-p} c^{n+2-q}$ and since $\le n$, $uv^0wx^0y$ contains less or equal number of $b$'s as comparison to $a$'s $uv^0wx^0y$ cannot belong to $L$. Hence, contradiction to pumping lemma.

We find that $vx$ cannot contain both $a$'s and $c$'s because the number of $b$'s between rightmost $a$ and leftmost $c$, and hence if we ta contain both $a$'s and $c$'s then condition $|vwx| \le n$ is not satisfied.

Hence, we conclude that we cannot have $vx$ such that $uv^i wx^i$ $L(i \ge 0)$. Therefore, it is not a CFL.

***Prob.48. Show that $L = \{a^n b^n c^n | n \ge 1\}$ is not context-free.***

***Sol.*** In every string of $L$ any symbol appears the same number as any other symbol. Also, $a$ cannot appear after $b$ and $c$ cannot appe $b$, and so on.

*(i) Step 1* – Assume $L$ is context-free. Let $n$ be the natural obtained by using pumping lemma.

*(ii) Step 2* – Let $u = a^n b^n c^n$. Then $|u| = 3n > n$. Write $u =$ where $|wy| \ge 1$, i.e., at least one of $w$ or $y$ is not $\Lambda$.

*(iii) Step 3* – $vwxyz = a^n b^n c^n$. As $1 \le |wy| \le n$, $w$ or $y$ cannot all the three symbols $a$, $b$, $c$. So, (a) $w$ or $y$ is of the form $a^i b^j$ (or $b^i c^j$) $i, j$ such that $i + j \le n$. Or (b) $w$ or $y$ is a string formed by the repetition one symbol among $a$, $b$, $c$.

When $w$ or $y$ is of the form $a^i b^j$, $w^2 = a^i b^j a^i b^j$ (or $y^2 = a^i b^j a^i b^j$). a substring of $vw^2xy^2z$, $vw^2xy^2z$ cannot be of the form $a^m b^n$ $vw^2xy^2z \notin L$.

When both $w$ and $y$ are formed by the repetition of a single symbol $v = a^i$ and $w = b^j$ for some $i$ and $j$, $i \le n, j \le n$), the string $vxz$ will con remaining symbol, say $a_1$.

Also, $a_1^n$ will be a substring of $vxz$ as $a_1$ does not occur in $w$ or number of occurrences of one of the other two symbols in $vxz$ in less and $n$ is the number of occurrences of $a_1$, So

$$vw^0xy^0z = vxz \notin L$$

Thus, for any choice of $w$ or $y$, we get a contradiction. ***Therefo not context-free.***

***Prob.49. Show that $L = \{a^p \mid p$ is prime$\}$ is not a context-free lan***

***(R.G.P.V., Dec.***

***Sol.*** We use the following property of $L$ – If $w \in L$, then $|w|$ is a $p$

**Step 1** – Suppose $L = L(G)$ is context-free. Let $n$ be the natural obtained by using the pumping lemma.



**Step 2** – Let $p$ be a prime number greater than $n$. Then $z = a^p \in L$. We write $z = uvwxy$.

**Step 3** – By pumping lemma, $uv^0wx^0y = uwy \in L$. So $|uwy|$ is a prime number, say $q$. Let $|ux| = r$. Then $|uv^q wx^q y| = q + qr$. As $q + qr$ is not a prime, $uv^q wx^q y \notin L$. This is a contradiction. Therefore, $L$ is not context-free.

***Prob.50. Explain pumping lemma for CFL's. Show that $L = \{a^n b^n c^n | n \ge 1\}$ is not a context free language.***

***(R.G.P.V., Dec. 2012)***

***Or***

***Explain pumping lemma for CFL. Prove that following language is CFL or not ?***

$$L = \{a^n b^n c^n | n \ge 1\}$$

***(R.G.P.V., Dec. 2013)***

***Sol.*** **Pumping Lemma for CFL** – Refer to Q.18.

Also, refer to Prob.48.

***Prob.51. Construct a CFG generating all integers (with sign).***

***(R.G.P.V., June 2010)***

***Sol.*** $G = (V, T, P, S)$

where, $V = \{S,$ <sign>, <digit>, <integer>$\}$

$T = \{ 0, 1, 2, 3, \ldots\ldots, 9, +, - \}$

$P$ consists of $S \rightarrow$ <sign> <integer>, <sign> $\rightarrow + \mid -$

<integer> $\rightarrow$ <digit> <integer> | <digit>

<digit > $\rightarrow 0|1|2|\ldots\ldots|9$

$L(G)$ = the set of all integers. For example, the derivation of –15 can be obtained as follows –

$S \Rightarrow$ <sign> <integer> $\Rightarrow -$ <integer>

$\Rightarrow -$ <digit> <integer> $\Rightarrow -1$ <integer> $\Rightarrow -1$ <digit>

$\Rightarrow -15.$

***Prob.52. Design context free grammars for the following languages – The set $\{0^n 1^n | n \ge 1\}$ i.e., the set of all strings of one or more 0's followed by an equal number of 1's.***

***(R.G.P.V., Dec. 2008)***

***Or***

***Write the CFG for the following language – $L = \{0^n 1^n | n \ge 1\}$***

***(R.G.P.V., Dec. 2013)***

***Sol.*** Required CFG can be defined as –

$$L(G) = \{(S), (0, 1), P, S\}$$

where p consists of

$$S \rightarrow 0S1|01$$

***Prob.53. Find context-free grammar for the following language*** ($n \geq 0, m \geq 0, k \geq 0$) –

$$L = \{a^n b^m c^k : n = m \text{ or } m \leq k\}$$ ***(R.G.P.V., June 200***

***Sol.*** We construct $G$ as follows –

$$G = (\{S, A, B\}, \{a, b, c\}, P, S)$$

where $P$ consists of

$$S \rightarrow aAbC|DbBc|bBc|aAb|abC|ab|\in$$
$$A \rightarrow aAb|ab$$
$$B \rightarrow bBc|cC|c$$
$$C \rightarrow cC|s$$
$$D \rightarrow aD|a$$

***Prob.54. Write CFG for the following sets –***

$$L = \{a^n\ b^m\ c^m\ a^n \mid n, m \geq 1\}$$ ***(R.G.P.V., Dec.***

***Sol.*** Required CFG can be defined as –

$$L(G) = \{(S, A), (a, b, c), P, S\}$$

where, $P$ consists of,

$$S \rightarrow aSa|aAa$$
$$A \rightarrow bAc|bc$$

***Prob.55. Write CFG for set of all words consisting of an equal no. of a's and b's.***

***For example : a a b b, a b a b, a b b b a a.*** ***(R.G.P.V., Dec.***

***Sol.*** Required CFG can be defined as follows –

$$L(G) = \{(S), (a, b), P, S\}$$

where $P$ consists of,

$$S \rightarrow aSbS|bSaS|\in$$

***Prob.56. Construct CFG for generating set –***

$$L = \{a^n b^{2n} c^m d^{3m} | n, m \geq 1\}$$ ***(R.G.P.V., Jun***

***Sol.*** The required *CFG* is defined as

$$G = (\{S, A, B\}, \{a, b, c, d\}, P, S)$$

where P consist of

$$S \rightarrow AB$$
$$A \rightarrow aAbb|abb$$
$$B \rightarrow cBddd|cddd$$

***Prob.57. Write a CFG to generate the language –***

$$L = \{0^m 1^n 0^{m+n} | m, n \geq 1\}.$$ ***(R.G.P.V., Jun***

***Sol.*** The productions of the grammar are –

$$S = 0A0$$
$$A \rightarrow 0A0|1B0|10$$
$$B \rightarrow 1B0|10$$



Because we want to get block of $m$ 0's, followed by a block of $n$ 1's, followed by a block of $m + n$ 0's generated, and it is required that $m$ and $n$ both are greater than or equal to 1. This requires that for every 0 in the first block, a corresponding 0 must be generated in the last block, as well as for every 1 in the second block, a corresponding 0 must be generated in the last block.

Therefore, the grammar $G$ is –

$$G = (\{S, A, B\}, \{0, 1\}, P, S)$$, where the members of $P$ are given above.

***Prob.58. Construct CFG for the following set –***

***(i)*** $\{a^{2n}\ bc \mid n \geq 1\}$

***(ii)*** $\{a^n b^m c^m d^m \mid m, n \geq 1\}$. ***(R.G.P.V., June 2007)***

***Sol.*** (i) $\{a^{2n}\ bc \mid n \geq 1\}$

We construct $G$ as follows –

$$G = (\{S, A\}, \{a, b, c\}, P, S\}$$

where, $P$ consists of

$$S \rightarrow aaAbc|aabc$$
$$A \rightarrow aaA|aa$$

(ii) $\{a^n\ b^m\ c^m\ d^n \mid m, n \geq 1\}$

We construct $G$ as follows –

$$G = \{(S, A), (a, b, c, d), P, S\}$$

The required productions are as follows –

$$S \rightarrow aSd|aAd$$
$$A \rightarrow bAc|bc$$

***Prob.59. Write CFG for the following sets –***

***(i)*** $L = \{a^n\ b^m\ c^k : n = m \text{ or } m \leq k, n \geq 0, m \geq 0, k \geq 0\}$.

***(ii)*** $L = \{a^{2n}\ bc \mid n \geq 1\}$. ***(R.G.P.V., Dec. 2009)***

***Sol.*** (i) Refer to Prob.53. (ii) Refer to Prob.58 (i).

***Prob.60. Construct a context free grammar for the languages –***

$L(G_1) = \{a^i b^{2i}, i > 0\}$ ***and*** $L(G_2) = (a^n b a^n, n > 0\}$ ***(R.G.P.V., Dec. 2010)***

***Sol.*** (i) $G_1 = (\{S\}, \{a, b\}, P, S)$

where $P$ consists of

$$S \rightarrow aSbb|abb$$

**Ans.**

(ii) Let, $G_2 = (\{S, A\}, (a, b), P, S)$, where, $P$ consists of $S \rightarrow aAa$, $A \rightarrow aAa|b$. This is the required grammar.

***Prob.61. Construct context free grammars to generate the following***

*(i)* $0^m 1^m$ $m \geq 0$

*(ii)* $0^m 1^n$ $1 \leq m \leq n$

*(iii)* $0^m 1^n 2^r$ $m = n$

*(iv)* $0^l 1^m 2^n$ $l + m = n$

***(R.G.P.V., Dec.***

***Sol.*** (i) Refer to Prob. 52, only difference is that $\in$ will also production S.

(ii) Required CFG can be defined as –

$$L(G) = \{(S), (0, 1), P, S\}$$

where P consists of

$$S \rightarrow 0S1 \mid 1 \mid \in$$

(iii) Refer to Prob. 53.

(iv) The productions of the grammar are –

$$S \rightarrow 0A2$$

$$A \rightarrow 0A2 \mid 1B2 \mid 12$$

$$B \rightarrow 1B2 \mid 12$$

Because we want to get block of $l$ 0's, followed by a block of followed by a block of $l + m$ 2's generated, and it is required that $l$ and are greater than or equal to 1. This requires that for every 0 in the first a corresponding 2 must be generated in the last block, as well as for ev the second block, a corresponding 2 must be generated in the last blo

$$G = (\{S, A, B\}, \{0, 1, 2\}, P, S)$$

where the productions P are given above.

***Prob.62. Find the CFG for the regular expression (110 + 11)*(0***

***(R.G.P.V., Dec.***

***Sol.*** Starting with grammars for the languages {0} and {1}; howe may simplify the process by taking some obvious shortcuts. The proce

$$A \rightarrow 110 \mid 11$$

generate the language {110, 11}. By grammar $G^*$ generating $L^*$, we c the productions

$$B \rightarrow AB \mid \Lambda$$

$$A \rightarrow 110 \mid 11$$

to generate the language {110, 11}*, using the start symbol $B$. Similarly, w

$$C \rightarrow DC \mid \Lambda$$

$$D \rightarrow 10$$

to derive {10}* from the start symbol $C$. Finally, we generate the conca of the two languages by adding the production $S \rightarrow BC$. The final g



as start symbol $S$, auxiliary variables $A$, $B$, $C$, and $D$ and productions

$$S \rightarrow BC$$

$$B \rightarrow AB \mid \Lambda$$

$$A \rightarrow 110 \mid 11$$

$$C \rightarrow DC \mid \Lambda$$

$$D \rightarrow 10$$

***Prob.63. Find the regular grammar for the language –***

***(i)*** $L = \{w \in \{a, b\}^* : w$ ***does not contain the substring*** $aa\}$

***(ii)*** $L = \{a^n b^n : n \geq 0\}$

***(R.G.P.V., Dec. 2010)***

***Sol.*** (i) Observe that whenever a $a$ occurs it must be followed immediately by a $b$. Such a substring may be preceded and followed by an arbitrary number of $b$'s. This suggests that the answer involves the repetition of strings of the form $b...bab...b$, that is, the language denoted by the regular expression $(b^*abb^*)^*$. However, the answer is still incomplete, since the strings ending in $a$ or consisting of all $b$'s are unaccounted for. After taking care of these special cases we arrive at the answer

$$r = (b^*abb^*)^* (a + \lambda) + b^* (a + \lambda)$$

If we see $L$ as the repetition of the strings $b$ and $ab$, the shorter expression will be –

$$r = (b + ab)^* (a + \lambda)$$

The DFA for this regular expression is as follows –

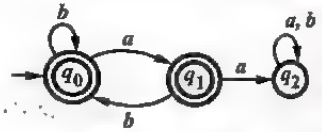



***Fig. 3.20 DFA***

Let $G = (\{A_0, A_1, A_2\}, \{a, b\}, P, A_0)$, where P is given by

$$\left.\begin{array}{l} q_0 \rightarrow aq_1 \\ q_0 \rightarrow a \\ q_0 \rightarrow bq_0 \\ q_0 \rightarrow b \end{array}\right\}$$

$$\left.\begin{array}{l} q_1 \rightarrow aq_2 \\ q_1 \rightarrow bq_0 \\ q_1 \rightarrow b \end{array}\right\}$$

$$\left.\begin{array}{l} q_2 \rightarrow aq_2 \\ q_2 \rightarrow bq_2 \end{array}\right\}$$

G is the required grammar.

***(ii) $L = \{a^n b^n : n \geq 0\}$***

Refer to Prob.30 (ii), Unit-II.

(Assume $a$ as 0 and $b$ as 1)

Since the language $L = \{a^n b^n : n \geq 0\}$ is not regular, so we can't ge its regular grammer.

***Prob.64. Given a grammar, make non-deterministic finite a and convert it to deterministic form –***

$$S \rightarrow 0S \mid 1A \mid 1$$
$$A \rightarrow 0 \mid 0A \mid 1S$$

***(R.G.P.V., June***

***Sol.*** Let $M = \{\{q_0, q_1, q_f\}, \{0, 1\}, \delta, q_0, \{q_f\}\}$, where $q_0$ and corresponding to $S$ and $A$, respectively, and $q_f$ is the final state.

$S \rightarrow 0S$ production induces a transition from $q_0$ to $q_0$ with label 0. Sim $S \rightarrow 1A$, $A \rightarrow 0A$, and $A \rightarrow 1S$ induce transitions from $q_0$ to $q_1$ with la from $q_1$ to $q_1$ with label 0, and from $q_1$ to $q_0$ with label 1, respect $S \rightarrow 1$ and $A \rightarrow 0$ induce transitions from $q_0$ to $q_f$ with label 1 and from $q_1$ to $q_f$ with label 0, respectively. Transition graph for this solution is given in fig. 3.21.

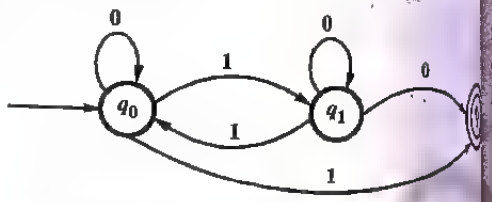



***Fig. 3.21 Transition Diagram of NDFA for G***

***Example***

Transition table correspond to fig. 3.21 is given in table 3.1.

**Table 3.1 Transition Table of NDFA**

| State | Input | |
|---|---|---|
| | 0 | 1 |
| $\rightarrow q_0$ | $q_0$ | $q_1, q_f$ |
| $q_1$ | $q_1, q_f$ | $q_0$ |
| $(q_f)$ | – | – |

**Table 3.2 Transition Table of**

| State | Input | |
|---|---|---|
| | 0 | 1 |
| $[q_0]$ | $[q_0]$ | $[q_1$ |
| $[q_1, q_f]$ | $[q_1, q_f]$ | $q_0$ |

Transition table of DFA can be constructed as given in table 3.2.

Transition diagram correspond to table 3.2 is drawn in fig. 3.22.

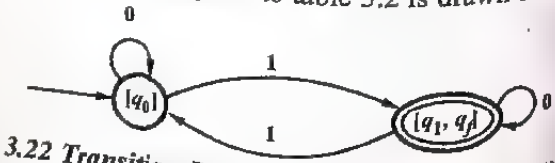



***Fig. 3.22 Transition Diagram of DFA for Given Example***



***Prob.65. Construct the context-free grammar (CFG) equivalent to a regular expression –***

$$(011 + 1)^* (01)^*$$ ***(R.G.P.V., Dec. 2005)***

***Or***

***Give CFG for R.E. $(011 + 1)^* (01)^*$.*** ***(R.G.P.V., June 2016)***

***Sol.*** Starting with grammars for the languages {0} and {1}; however, we may simplify the process by taking some obvious shortcuts. The productions

$$A \rightarrow 011 \mid 1$$

generate the language {011, 1}. By grammar $G^*$ generating $L^*$, we can use the productions

$$B \rightarrow AB|\Lambda$$
$$A \rightarrow 011 \mid 1$$

to generate the language {011, 1}*, using the start symbol $B$. Similarly, we can use

$$C \rightarrow DC \mid \Lambda$$
$$D \rightarrow 01$$

to derive {01}* from the start symbol $C$. Finally, we generate the concatenation of the two languages by adding the production $S \rightarrow BC$. The final grammar has start symbol $S$, auxiliary variables $A$, $B$, $C$, and $D$ and productions

$$S \rightarrow BC$$
$$B \rightarrow AB \mid \Lambda$$
$$A \rightarrow 011 \mid 1$$
$$C \rightarrow DC \mid \Lambda$$
$$D \rightarrow 01$$

***Prob.66. Find L(G) for the grammer***

$$S \rightarrow aCa$$
$$C \rightarrow aCa|b$$ ***(R.G.P.V., Dec. 2016***

***Sol.*** Required $L(G)$ can be defined as

$L(G) = \{(S, C), (a, b), P, S\}$

Here $P$ consist of

$$S \rightarrow aCa$$
$$C \rightarrow aCa|b$$

and also

$$L = \{a^n b a^n | n \geq 1\}$$

***Prob.67. Show that the following two grammars are equivalent –***

| ***Grammar-1*** | ***Grammar-2*** |
|---|---|
| $S \rightarrow abAB\|ba$ | $S \rightarrow abAaA\|abAbb\|ba$ |
| $A \rightarrow aaa$ | $A \rightarrow aaa$ |
| $B \rightarrow aA\|bb$ | |

***(R.G.P.V., Dec. 2009)***

***Sol.*** If the strings generated by both the grammars are the same, then we can say that both the grammars are equivalent.

First, we check strings generated by the starting symbol $S$ of grammar-1.

The strings are –

$S \Rightarrow abAB \Rightarrow abaaaB \Rightarrow abaaaaA \Rightarrow abaaaaaaa$

$S \Rightarrow abAB \Rightarrow abaaaB \Rightarrow abaaabb$

$S \Rightarrow ba$

Now, we check strings generated by the starting symbol $S$ of grammar-2.

The strings are –

$S \Rightarrow abAaA \Rightarrow abaaaaaaa$

$S \Rightarrow abAbb \Rightarrow abaaabb$

$S \Rightarrow ba$

Therefore, grammar-1 and grammar-2 are equivalent.

***Prob.68. Consider the grammar with productions –***

$$S \rightarrow aaB,\ A \rightarrow bBb \mid \lambda,\ B \rightarrow Aa$$

***Show that the string aabbabba is not in the language generated by grammar.*** ***(R.G.P.V., June 2***

***Sol.*** $S \Rightarrow aaB \Rightarrow aaAa \Rightarrow aabBba$

$\Rightarrow aabAaba \Rightarrow aabbBbaba$

$\Rightarrow aabbAababa \Rightarrow aabbababa$

Since, the given string is *aabbabba*.

Hence, the string *aabbabba* is not in the language.

# UNIT 4

## INTRODUCTION OF PDA, FORMAL DEFINITION, CLOSURE PROPERTY OF PDA, EXAMPLES OF PDA, DETERMINISTIC PUSHDOWN AUTOMATA, NPDA

***Q.1. Write short note on pushdown automata.***

***Ans.*** Pushdown Automata (PDA) is just like non-deterministic finite automata with an extra memory component called stack. The stack allows pushdown automata to accept or recognize those languages which are beyond the limit of finite automata.

The classes of language accepted by the PDA is exactly class of context-free grammar (i.e. the language acceptance power of PDA is same as that of context-free grammar). This equivalence is very useful because it provides another option by which we can justify that a language is context-free (either generate its context-free grammar or create a pushdown automata which recognise it). The model of pushdown automata is shown in fig. 4.1.

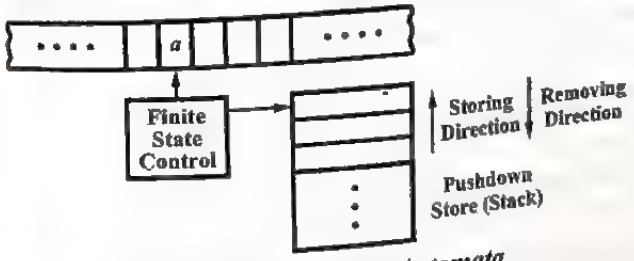



***Fig. 4.1 Pushdown Automata***

***Q.2. Give the definition of pushdown automata with the help of diagram.*** ***(R.G.P.V., Dec. 2014)***

***Or*** ***(R.G.P.V., June 2016)***

***Explain PDA.***

***Or*** ***(R.G.P.V., Dec. 2016)***

***Give the formal definition of PDA.***

**Ans.** A ***pushdown automaton*** is a 7-tuple machine, viz. $(Q, \Sigma, \Gamma, Z_0, F)$ consists of –

(i) A finite non-empty set of states denoted by $Q$.

(ii) A finite non-empty set of input symbols denoted by $\Sigma$.

(iii) A finite non-empty set of pushdown symbols denoted by

(iv) A special state called the initial state denoted by $q_0$.

(v) A special pushdown symbol called the initial symbol pushdown stack denoted by $Z_0$.

(vi) A set of final states, which is a subset of $Q$ denoted by

(vii) The transition function $\delta$ from $Q \times (\Sigma \cup \{\Lambda\}) \times \Gamma$ to the finite subsets of $Q \times \Gamma^*$.

***Q.3. What is the additional feature PDA has when compared with***

***(R.G.P.V., June***

**Ans.** Pushdown automata are like non-deterministic finite automa have an extra component called a stack. The stack provides additional m beyond the finite amount available in the control. The stack allows pus automata to recognize some non-regular languages.

***Q.4. Differentiate between "PDA by empty stack" and "PDA by state".***

***(R.G.P.V., Dec.***

**Ans.** If the language accepted by a pushdown automaton to be the all inputs for which some sequence of moves causes the pushdown auto to empty its stack then this language is referred to as the language accep the empty stack. And, the PDA is called "PDA by empty stack".

If the language accepted by a pushdown automaton to be the set inputs for which some choices of moves causes the pushdown automa enter a final state, then this language is referred to as the language acc by the final state. And, the PDA is called "PDA by final state".

***Q.5. State the difference between PDA and the FA. (R.G.P.V., Dec.***

**Ans.** The difference between a PDA and an FA is the length of the formed by a given input. If the string of seven letters is fed into an FA, it fo a path exactly seven edges long. In a PDA, the path could be longer or s

***Q.6. What is PDA ? Explain instantaneous description of PDA.***

***(R.G.P.V., Dec.***

**Ans.** Refer to Q.2.

The triplet $(q, w, u)$, where $q$ is the state of the control unit, $w$ remaining unread part of the input string, and $u$ is the stack contents (w leftmost symbol indicating the top of the stack) is called an instanta description (ID) of a pushdown automaton.



A move from one instantaneous description to another will be denoted by the symbol '$\vdash$', thus

$$(q_1, aw, bx) \vdash (q_2, w, yx)$$

is possible if and only if

$$(q_2, y) \in \delta(q_1, a, b).$$

Moves containing an arbitrary number of steps will be denoted by $\vdash^*$. On occasions where several automata are under consideration, we will use $\vdash_M$ to show that the move is made by the particular automaton $M$.

It must be noted that an initial ID is $(q_0, w, Z_0)$. It means initially the PDA is in the initial state $q_0$, the input string to be processed is $w$, and the stack has only one symbol, viz. $Z_0$. Also in an ID $(q, w, u)$, $w$ may be $\lambda$. In this case the PDA makes a $\lambda$-move.

***Q.7. Define the language accepted by a pushdown automaton. Also, define the set N(A) accepted by null store.***

***Or***

***Explain how many ways PDA can accept (final out null store).***

***(R.G.P.V., June 2016)***

**Ans.** Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a non-deterministic pushdown automaton. The language accepted by $M$ is the set

$$L(M) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash^*_M (q_f, \lambda, \alpha)$$
$$\text{for some } q_f \in F \text{ and } \alpha \in \Gamma^*\}$$

In words, the languages accepted by $M$ is the set of all strings that can put $M$ into a final state at the end of the string. The stack content $\alpha$ is irrelevant to this definition of acceptance. Let us take an example.

Construct an NPDA for the language

$$L = \{w \in \{a, b\}^* \mid n_a(w) = n_b(w)\}$$

The solution to this problem involves counting the number of $a$'s and $b$'s, which is easily done with a stack. We need not even worry about the order of the $a$'s and $b$'s. We can insert a counter symbol, say 0, into the stack whenever an $a$ is read, then pop one counter symbol from the stack when a $b$ is encountered. The only problem with this is that if there is a prefix of $w$ with more $b$'s than $a$'s, we will not find a 0 to use. But, this is easy to fix. For this, we can use a negative counter symbol, say 1, for counting the $b$'s that are to be matched against $a$'s later. The complete solution is an NPDA.

$M = (\{q_0, q_f\}, \{a, b\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_f\})$ with $\delta$ given as –

$$\delta(q_0, \lambda, Z_0) = \{(q_f, Z_0)\}$$
$$\delta(q_0, a, Z_0) = \{(q_0, 0Z_0)\}$$
$$\delta(q_0, b, Z_0) = \{(q_0, 1Z_0)\}$$

$$\delta(q_0, a, 0) = \{(q_0, 00)\}$$
$$\delta(q_0, b, 0) = \{(q_0, \lambda)\}$$
$$\delta(q_0, a, 1) = \{(q_0, \lambda)\}$$
$$\delta(q_0, b, 1) = \{(q_0, 11)\}$$

In processing the string *baab*, the NPDA makes the moves

$$\begin{aligned}(q_0, baab, Z_0) &\vdash (q_0, aab, 1Z_0)\\ &\vdash (q_0, ab, Z_0)\\ &\vdash (q_0, b, 0Z_0)\\ &\vdash (q_0, \lambda, Z_0)\\ &\vdash (q_f, \lambda, Z_0)\end{aligned}$$

and hence the string is accepted.

The second type of acceptance is defined as –

Let $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. The set $N(A)$ accepted by store (or empty store) is defined by

$$N(A) = \{w \in \Sigma^* | (q_0, w, Z_0) \vdash^* (q, \lambda, \lambda) \text{ for some } q \in Q\}$$

In other words, $w$ is in $N(A)$ if $A$ is in initial ID $(q_0, w, Z_0)$ and empties the after processing all the symbols of $w$. So in defining $N(A)$, we consider the c brought about on the stack by application of $w$, and not the transition of stat

***Q.8. Write a brief note on 2-way PDA.*** ***(R.G.P.V., Dec.***

***Ans.*** A two-way pushdown automaton or 2-PDA is a pushdown autom that is permitted to move in either way on its input. Like the 2-way automaton (2FA), it accepts by moving off the right end of its input in state. The language $L = \{0^n1^n2^n \mid n \geq 1\}$ is accepted by a two-way push automaton. The $L$ is not a context-free language, by the way, so 2-PDA not equivalent to PDA's.

Any language accepted by a 2DPDA is recognizable in linear time computer. Thus, the existence of a context-free language requiring more linear time to recognize on a computer, would imply that there are co free languages not accepted by 2DPDAs. However, no one till now has p that such a language exists. Incidentally, the language $\{0^n1^n2^n \mid n \geq 1\}$ example of a non context-free language accepted by a 2DPDA.

***Q.9. Prove that, if $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is a PDA accepting empty store, we can find a PDA***

$$B = (Q', \Sigma, \Gamma', \delta_B, q_0', Z_0', F')$$

***which accepts L by final state, i.e., $L = N(A) = T(B)$.***

***Ans.*** $B$ is constructed in such a way that – (i) by the initial move reaches an initial ID of $A$, (ii) by the final move of $B$, it reaches its final and (iii) all intermediate moves of $B$ are as in $A$.

Let us define $B$ as follows –

$$B = (Q', \Sigma, \Gamma, \delta_B, q_0', Z_0', F)$$

where, $q_0'$ is a new state (not in $Q$).
$F' = \{q_f\}$, with $q_f$ as a new state (not in $Q$),
$Q' = Q \cup \{q_0', q_f\}$.
$Z_0'$ is a new start symbol for PDS of $B$,
$\Gamma' = \Gamma \cup (Z_0')$, and
$\delta_B$ is given by the rules $R_1, R_2, R_3$

with

$$R_1 : \delta_B(q_0', \lambda, Z_0') = \{(q_0, Z_0, Z_0')\},$$
$$R_2 : \delta_B(q, a, Z) = \delta(q, a, Z) \text{ for all } (q, a, Z) \text{ in } Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$$
$$R_3 : \delta_B(q, \lambda, Z_0') = \{(q_f, \lambda)\} \qquad \text{for all } q \in Q$$

By $R_1$, the PDA $B$ moves from an initial ID of $B$ to an initial ID of $A$. $R_1$ gives a $\lambda$-move. As a result of $R_1$, $B$ moves to the initial state of $A$ with the start symbol $Z_0$ on the top of PDS.

$R_2$ is used to simulate $A$. Once $B$ reaches an initial ID of $A$, $R_2$ can be used to simulate moves of $A$. We can repeatedly apply $R_2$ until $Z_0'$ is pushed to the top of PDS. As $Z_0'$ is a new pushdown symbol, we have to use $R_3$.

$R_3$ gives a $\lambda$-move. Using $R_3$, $B$ moves to the new (final) state $q_f$ erasing $Z_0'$ in PDS.

Thus the behaviour of $B$ and $A$ are similar except for the $\lambda$-moves given by $R_1$ and $R_3$. Also, $w \in T(B)$ if and only if $B$ reaches $q_f$, i.e., if and only if the PDS has no symbols from $\Gamma$ (since $B$ can reach $q_f$ only by the application of $R_3$). This suggests that $T(B) = N(A)$.

Now we prove rigorously that $N(A) = T(B)$. Suppose $w \in N(A)$. Then by the definition of $N(A)$, $(q_0, w, Z_0) \overset{*}{\underset{A}{\vdash}} (q, \lambda, \lambda)$ for some $q \in Q$. Using $R_2$ we see that

$$(q_0, w, Z_0) \overset{*}{\underset{B}{\vdash}} (q, \lambda, \lambda)$$

By result 2,

$$(q_0, w, Z_0 Z_0') \overset{*}{\underset{B}{\vdash}} (q, \lambda, Z_0') \qquad \text{....(i)}$$

By $R_1$, $(q_0', \lambda, Z_0') \overset{}{\underset{B}{\vdash}} (q_0, \lambda, Z_0 Z_0')$

By result 1, we have

$$(q_0', w, Z_0') \overset{}{\underset{B}{\vdash}} (q_0, w, Z_0 Z_0') \qquad \text{....(ii)}$$

By $R_3$, $(q, \lambda, Z_0') \overset{}{\underset{B}{\vdash}} (q_f, \lambda, \lambda)$ ....(iii)

Combining equation (i) – equation (iii), we have

$$(q_0', w, Z_0') \overset{*}{\underset{B}{\vdash}} (q_f, \lambda, \lambda)$$

This proves that $w \in T(B)$, i.e., $N(A) \subseteq T(B)$.

To prove $T(B) \subseteq N(A)$, start with $w \in T(B)$. Then

$$(q'_0, w, Z'_0) \overset{*}{\underset{B}{\vdash}} (q_f, \lambda, \alpha)$$

But $B$ can reach $q_f$ only by application of $R_3$. To apply $R_3$, $Z_0$ sh... the top most element on PDS. $Z'_0$ is placed initially, and so when it is... top there are no other elements in PDS. So $\alpha = \lambda$ and equation (iv) ... reduces to

$$(q'_0, w, Z'_0) \overset{*}{\underset{B}{\vdash}} (q_f, \lambda, \lambda)$$

In equation (v), the initial and final steps are effected only by ... The intermediate steps are induced by the corresponding moves of ... equation (v) can be split as

$$(q'_0, \lambda w, Z'_0) \underset{B}{\vdash} (q_0, w, Z_0 Z'_0) \overset{*}{\underset{B}{\vdash}} (q, \lambda, Z'_0)$$

for some $q \in Q$. Thus, $(q'_0, \lambda w, Z'_0) \underset{B}{\vdash} (q_0, w, Z_0 Z'_0) \overset{*}{\underset{B}{\vdash}} (q, \lambda, Z...$ $(q_f, \lambda, \lambda)$. As we get $(q_0, w, Z_0 Z'_0) \overset{*}{\underset{B}{\vdash}} (q_1, \lambda, Z'_0)$ by applying $R_1$ ... times and $R_2$ does not affect $Z'_0$ at the bottom, we have $(q_0, w, Z_0) \overset{*}{\underset{B}{\vdash}} ...$ $\lambda)$. By the construction of $R_2$, $(q_0, w, Z_0) \overset{*}{\underset{A}{\vdash}} (q, \lambda, \lambda)$, which means ... Thus, $T(B) \subseteq N(A)$, and hence $T(B) = N(A) = L$.

***Q.10. Write short note on deterministic pushdown automata.*** ***(R.G.P.V., Dec. ...***

***Or***

***Define deterministic PDA.*** ***(R.G.P.V., Dec. ...***

***Ans.*** Refer to Q.9.

***Q.11. Explain closure properties of PDA.***

***Ans.*** The power of pushdown automata (PDA) and context-free g... are equivalent. So, the closure properties of PDA are also equivalent to ... properties of context-free grammar.

Refer to Q.21 (Unit-III).

***Q.12. Write short note on NPDA and DPDA.*** ***(R.G.P.V., Dec. ...***

***Or***

***Explain the difference between deterministic PDA and ... deterministic PDA with example.*** ***(R.G.P.V., June ...***

***Ans.*** A ***non-deterministic pushdown accepter*** (NPDA) is defined ... seven tuple –

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$$



where, $Q$ is a finite set of internal states of the control unit (i.e., finite state control)

$\Sigma$ is the input alphabet

$\Gamma$ is a finite set of symbols called stack alphabet

$\delta$: $Q \times (\Sigma \cup \{\lambda\} \times \Gamma) \rightarrow$ finite subsets of $Q \times \Gamma^*$ is the transition function

$q_0 \in Q$ is the initial state of control unit

$z \in \Gamma$ is the stack start symbol

$F \subseteq Q$ is the set of final states.

***Deterministic pushdown acceptor automaton (DPDA)*** is a pushdown automaton that never has a choice in its move.

A pushdown automaton $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ is said to be deterministic if it is an NPDA subject to the restrictions that, for every $q \in Q$, $a \in \Sigma \cup \{\lambda\}$ and $b \in \Gamma$

(i) $\delta(q, a, b)$ contains at most one element.

(ii) if $\delta(q, \lambda, b)$ is not empty then $\delta(q, c, b)$ must be empty for every $c \in \Sigma$

The first of these conditions simply requires that for any given input symbol and any stack top at most one move can be made. The second condition is that when a $\lambda$-move is possible for some configuration, no input-consuming alternative is available.

***Q.13. Is it true that non-deterministic PDA is more powerful than that of deterministic PDA ? Justify your answer.*** ***(R.G.P.V., Dec. 2014)***

***Ans.*** Yes, it is true that non-deterministic PDA is more powerful than that of deterministic PDA. A pushdown automaton that has the property of mapping from input symbol, state and stack symbol can result in more than one next state and/or stack state. A non-deterministic PDA can be used to parse a context-sensitive grammar and is consequently more powerful than a DPDA. The languages accepted by NPDA is large than that of the DPDA.

***Q.14. Deterministic and non-deterministic models of PDA are not equivalent. Justify with example.*** ***(R.G.P.V., June 2010)***

***Or***

***Is it true that deterministic PDA and non-deterministic PDA are equivalent in the sense of language of acceptances ? Justify your answer.*** ***(R.G.P.V., June 2015)***

***Ans.*** For finite automata, the deterministic and non-deterministic models are equivalent with respect to the languages accepted. The same is not true ... PDA. In fact $ww^R$ is accepted by a non-deterministic PDA, but not by ... deterministic PDA.

For example – The following fig. 4.2, gives a PDA that accepts $\{ww^R \mid w \text{ in } (0+1)^*\}$. Rules (i) through (vi) allow $M$ to store the input on the stack. In rules (iii) and (vi), $M$ has a choice of two moves. $M$ may decide that the middle of the input string has been reached and make the second choice – $M$ goes to state $q_2$ and tries to match the remaining input symbols with the contents of the stack. If $M$ guessed right, and if the input is of the form $ww^R$, then the inputs will match, $M$ will empty its stack and thus accept the input string.

Like the non-deterministic finite automaton, a non-deterministic PDA, $M$ accepts an input if any sequence of choices cause $M$ to empty its stack. Thus $M$ always "guesses right" because wrong guesses, in themselve, do not cause an input to be rejected. An input is rejected only if there is no "right guess". Fig. 4.3 shows the accessible ID's of $M$ when processes the string 001100.

$M = (\{q_1, q_2\}, \{0, 1\}, \{R, B, G\}, \delta, q_1, R, \phi)$

(i) $\delta(q_1, 0, R) = \{(q_1, BR)\}$
(ii) $\delta(q_1, 1, R) = \{(q_1, GR)\}$
(iii) $\delta(q_1, 0, B) = \{(q_1, BB), (q_2, \in)\}$
(iv) $\delta(q_1, 0, G) = \{(q_1, BG)\}$
(v) $\delta(q_1, 1, B) = \{(q_1, GB)\}$
(vi) $\delta(q_1, 1, G) = \{(q_1, GG), (q_2, \in)\}$
(vii) $\delta(q_2, 0, B) = \{(q_2, \in)\}$
(viii) $\delta(q_2, 1, G) = \{(q_2, \in)\}$
(ix) $\delta(q_1, \in, R) = \{(q_2, \in)\}$
(x) $\delta(q_2, \in, R) = \{(q_2, \in)\}$

***Fig. 4.2 A Non-deterministic PDA accepts $\{ww^R \mid w \text{ in } (0+1)^*\}$ by Empty Stack***

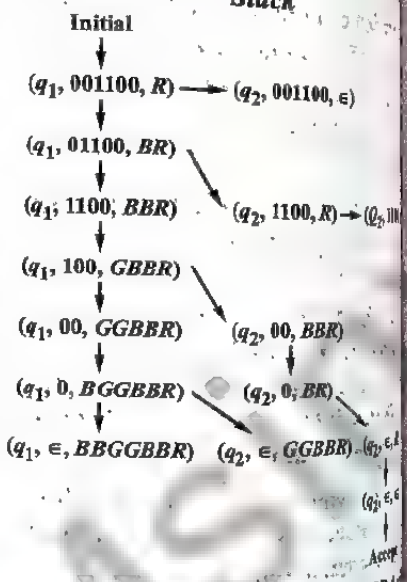



***Fig. 4.3 Accessible ID's for the PDA Input 001100***

## NUMERICAL PROBLEMS

***Prob.1. Construct the PDA that accepts $\{wcw^R \mid w \in (0+1)^*\}$ by (...) stack, where $w^R$ represents the reverse of $w$.*** ***(R.G.P.V., June 20...)***

***Sol.*** We define the PDA $A$ as follows –

$A = (\{q_0, q_1, q_f\}, \{0, 1, c\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_f\})$ where $\delta$ is defined as –

$\delta(q_0, 0, Z_0) = \{(q_0, 0Z_0)\}$
$\delta(q_0, 1, Z_0) = \{(q_0, 1Z_0)\}$



$\delta(q_0, 0, 0) = \{(q_0, 00)\}$
$\delta(q_0, 1, 0) = \{(q_0, 10)\}$
$\delta(q_0, 0, 1) = \{(q_0, 01)\}$
$\delta(q_0, 1, 1) = \{(q_0, 11)\}$
$\delta(q_0, c, 0) = \{(q_1, 0)\}$
$\delta(q_0, c, 1) = \{(q_1, 1)\}$
$\delta(q_0, c, Z_0) = \{(q_1, Z_0)\}$
$\delta(q_1, 0, 0) = \{(q_1, \lambda)\}$
$\delta(q_1, 1, 1) = \{(q_1, \lambda)\}$
$\delta(q_1, \lambda, Z_0) = \{(q_f, Z_0)\}$

We observe that from the construction of $A$, these rules cannot erase $Z_0$. For erasing $Z_0$ from the stack we consider an additional rule –

$\delta(q_f, \lambda, Z_0) = \{(q_f, \lambda)\}$

By this rule stack can be emptied by $\lambda$-moves iff the PDA reaches the ID $(q_f, \lambda, Z_0)$. Hence,

$$N(A) = \{wcw^R | w \in \{0, 1\}^*\}.$$

***Prob.2. Construct PDA that accepts language***

$$L = \{WW^R \mid W \text{ in } (0+1)^*\}.$$ ***(R.G.P.V., Dec. 2012)***

***Or***

***Design PDA to accept the language $L(G) = \{WW^R / W \in (0, 1)^*$ and $W^R$ is the reverse of word $W\}$.*** ***(R.G.P.V., Dec. 2016)***

***Or***

***Design PDA to accept $\{WW^R / W \in (0, 1)^*\}$, where $W$ is a word and $W^R$ is reverse of word.*** ***(R.G.P.V., Nov. 2018)***

***Sol.*** We define the PDA $A$ as follows –

$$A = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

where $\delta$ is defined as –

$\delta(q_0, 0, Z_0) = \{(q_0, 0Z_0)\}$
$\delta(q_0, 1, Z_0) = \{(q_0, 1Z_0)\}$
$\delta(q_0, 0, 0) = \{(q_0, 00)\}$
$\delta(q_0, 1, 0) = \{(q_0, 10)\}$
$\delta(q_0, 0, 1) = \{(q_0, 01)\}$
$\delta(q_0, 1, 1) = \{(q_0, 11)\}$
$\delta(q_0, \wedge, 0) = \{(q_1, 0)\}$
$\delta(q_0, \wedge, 1) = \{(q_1, 1)\}$
$\delta(q_0, \wedge, Z_0) = \{(q_1, Z_0)\}$
$\delta(q_1, 0, 0) = \{(q_1, \wedge)\}$
$\delta(q_1, 1, 1) = \{(q_1, \wedge)\}$

$$\delta(q_1, \wedge, Z_0) = \{(q_2, Z_0)\}$$

Thus, $N(A) = \{WW^R | W \text{ in } (0+1)^*\}$

***Prob.3. Construct PDA for the set $L = \{a^n b^{2n} \mid n \geq 1\}$.***

***(R.G.P.V., June 2008, Dec. 20***

***Or***

***Design pushdown automata which accepts $L = \{0^n 1^{2n} \mid n \geq 1\}$.***

***(R.G.P.V., June 20***

***Sol.*** The required PDA $A$ is defined as follows –

$A = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, Z_0\}, \delta, q_0, Z_0, \phi)$, where $\delta$ is defined

$$\delta(q_0, a, Z_0) = \{(q_1, aZ_0)\}$$
$$\delta(q_1, a, a) = \{(q_1, aa)\}$$
$$\delta(q_1, b, a) = \{(q_2, a)\}$$
$$\delta(q_2, b, a) = \{(q_1, \lambda)\}$$
$$\delta(q_1, \lambda, Z_0) = \{(q_1, \lambda)\}$$

Let us see the acceptability for the string *aabbbb*. The sequence of mo in accepting *aabbbb* is

$$\begin{aligned}(q_0, aabbbb, Z_0) &\vdash (q_1, abbbb, aZ_0)\\ &\vdash (q_1, bbbb, aaZ_0)\\ &\vdash (q_2, bbb, aaZ_0)\\ &\vdash (q_1, bb, aZ_0)\\ &\vdash (q_2, b, aZ_0)\\ &\vdash (q_1, \lambda, Z_0)\\ &\vdash (q_1, \lambda, \lambda)\end{aligned}$$

Thus, $N(A) = \{a^n b^{2n} | n \geq 1\}$

***Prob.4. Construct PDA for the set $L = \{a^{2n} b^n | n \geq 1\}$.***

***(R.G.P.V., Dec. 2003, 2010, June 20***

***Sol.*** The required PDA $A$ is defined as follows –

$A = (\{q_0, q_1\}, \{a, b\}, \{a, Z_0\}, \delta, q_0, Z_0, \phi)$, where $\delta$ is defined by

$$\delta(q_0, a, Z_0) = \{(q_1, aZ_0)\}$$
$$\delta(q_0, a, a) = \{(q_1, aa)\}$$
$$\delta(q_1, a, a) = \{(q_0, a)\}$$
$$\delta(q_0, b, a) = \{(q_0, \lambda)\}$$
$$\delta(q_0, \lambda, Z_0) = \{(q_0, \lambda)\}$$

Let us see the acceptability for the string *aaaabb*. The sequence of mo in accepting the string is –

$$(q_0, aaaabb, Z_0) \vdash (q_1, aaabb, aZ_0)$$



$$\begin{aligned}&\vdash (q_0, aabb, aZ_0)\\ &\vdash (q_1, abb, aaZ_0)\\ &\vdash (q_0, bb, aaZ_0)\\ &\vdash (q_0, b, aZ_0)\\ &\vdash (q_0, \lambda, Z_0)\\ &\vdash (q_0, \lambda, \lambda)\end{aligned}$$

Thus, $N(A) = \{a^{2n} b^n | n \geq 1\}$

***Prob.5. Construct a PDA A accepting the set of all strings over {a, b} having equal number of a's and b's.***

***Or***

***Design a PDA which accepts the language L = {W ∈(a, b)*/W has the equal number of a's and b's}.*** ***(R.G.P.V., Dec. 2013)***

***Sol.*** Let $A = (\{q\}, \{a,b\}, \{Z_0, a, b\}, \delta, q_0, Z_0, \phi)$

where $\delta$ is defined by the following rules –

$$\delta(q,a,Z_0) = \{(q,aZ_0)\}\ \delta(q,b,Z_0) = \{(q,bZ_0)\}$$
$$\delta(q,a,a) = \{(q,aa)\}\ \delta(q,b,b) = \{(q,bb)\}$$
$$\delta(q,a,b) = \{(q,\lambda)\}\ \delta(q,b,a) = \{(q,\lambda)\}$$
$$\delta(q,\lambda,Z_0) = \{(q,\lambda)\}$$

We begin with storing a symbol of the input string and continue storing until the other symbol occurs. If the topmost symbol in the stack is $a$ and the current input symbol is $b$, $a$ in the stack is erased. If $u$ has equal number of $a$'s and $b$'s, then $(q, u, Z_0) \vdash^* (q, \lambda, Z_0) \vdash (q, \lambda, \lambda)$, So $u \in N(A)$. We can show that $N(A)$ is the given set of strings over $\{a,b\}$ using the construction of $\delta$. Thus, $A$ is the required PDA.

***Prob.6. Construct PDA for the following language –***

$$L = \{a^m b^n c^{m+n} \mid m, n \geq 1\}.$$

***(R.G.P.V., Dec. 2006, 2014, June 2015)***

***Sol.*** The PDA accepting $\{a^m b^n c^{m+n} \mid m, n \geq 1\}$ is defined as follows –

$A = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{Z_0, Z_1, Z_2\}, \delta, q_0, Z_0, \phi)$

where $\delta$ is given by

$$\delta(q_0, a, Z_0) = \{(q_1, Z_1Z_0)\}$$
$$\delta(q_0, a, Z_1) = \{(q_1, Z_1Z_1)\}$$
$$\delta(q_1, b, Z_1) = \{(q_1, Z_2Z_1)\}$$
$$\delta(q_1, b, Z_2) = \{(q_1, Z_2Z_2)\}$$
$$\delta(q_1, c, Z_2) = \{(q_2, \in)\}$$
$$\delta(q_2, c, Z_2) = \{(q_2, \in)\}$$

$$\delta(q_2, c, Z_1) = \{(q_2, \in)\}$$
$$\delta(q_2, \in, Z_0) = \{(q_2, \in)\}$$

***Prob.7. Construct PDA for the following language –***

$$L = \{a\, b^n\, c\, d^n \mid n \geq 1\}.$$

***(R.G.P.V., June***

***Sol.*** The PDA accepting $\{ab^n\, cd^n \mid n \geq 1\}$ is defined as follows –

$$A = (\{q_0, q_1, q_2\}, \{a, b, c, d\}, \{b, Z_0\}, \delta, q_0,$$

where $\delta$ is given by

$$\delta(q_0, a, Z_0) = \{(q_1, Z_0)\}$$
$$\delta(q_1, b, Z_0) = \{(q_1, bZ_0)\}$$
$$\delta(q_1, b, b) = \{(q_1, bb)\}$$
$$\delta(q_1, c, b) = \{(q_2, b)\}$$
$$\delta(q_2, d, b) = \{(q_2, \in)\}$$
$$\delta(q_2, \in, Z_0) = \{(q_2, \in)\}$$

***Prob.8. Design a PDA to accept the following language –***
$$\{0^n 1^n \mid n \geq 1\}.$$
***(R.G.P.V., Dec.***

***Sol.*** **Pushdown Automata** – Refer to Q.1.

Let $A = \{q_0, q_1\}, \{0, 1\}, \{0, Z_0\}, \delta, q_0, Z_0, \phi)$

where $\delta$ is given by

$$\delta(q_0, 0, Z_0) = \{(q_0, 0Z_0)\}$$
$$\delta(q_0, 0, 0) = \{(q_0, 00)\}$$
$$\delta(q_0, 1, 0) = \{(q_1, \lambda)\}$$
$$\delta(q_1, 1, 0) = \{(q_1, \lambda)\}$$
$$\delta(q_1, \lambda, Z_0) = \{(q_1, \lambda)\}$$

The PDA will start in state $q_0$ and go on pushing 0's onto stack wi changing its state, until it gets 1. When it gets 1, it changes state to $q_1$ pops 0 from the top of the stack. Then for every 1 appearing next in the it simply pops 0 from the top of the stack, without changing a state. Wh reaches end of the input, if top of the stack is $Z_0$,then it empties its stack

***Prob.9. Construct a PDA for the following language –***

***(i)*** $a^n b^n \quad n \geq 0$ ***(ii)*** $a^n b^{2n} \quad n \geq 1$

***(R.G.P.V., Dec.***

***Sol.*** (i) Refer to Prob.8.

(ii) Refer to Prob.3.

***Prob.10. Construct PDA accepting*** $L = \{a^i b^j \mid j = i \text{ or } j = 2i\}$.

***(R.G.P.V., June***



***Sol.*** The logic that we use for the PDA to be constructed is – the PDA will start in state $q_0$, and either go on pushing the $a$'s onto the stack without changing its state, until it gets $b$, or will push two $a$'s first time when it gets $a$, in the input and change a state to $q_1$ and then go on pushing two $a$'s for every $a$ appearing next in the input remaining in state $q_1$, until it gets $b$. When it gets $b$ it pops $a$ from the top of the stack advances the tape head one position right and changes the state to $q_2$ irrespective of whether it is presently in state $q_0$ or $q_1$. Then for every $b$ appearing next in the input it simply pops a from the top of the stack, without changing a state. When it reaches end of the input, if top of the stack is $Z_0$, then it empties its stack. Therefore, the moves of the PDA are –

$\delta(q_0, \in, Z_0) = \{(q_2, \in)\}$ To take care of acceptance of empty string $\in$.

$$\delta(q_0, a, Z_0) = \{(q_0, aZ_0), (q_1, aaZ_0)\}$$
$$\delta(q_0, a, a) = \{(q_0, aa)\}$$
$$\delta(q_1, a, a) = \{(q_1, aaa)\}$$
$$\delta(q_0, b, a) = \{(q_2, \in)\}$$
$$\delta(q_1, b, a) = \{(q_2, \in)\}$$
$$\delta(q_2, b, a) = \{(q_2, \in)\}$$
$$\delta(q_2, \in, Z_0) = \{(q_2, \in)\}$$

Therefore, PDA is –

$$M = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, Z_0\}, \delta, q_0, Z_0, \phi)$$

where $\delta$ is given above.

***Prob.11. Construct a PDA for accepting the language***

$$L = \{ww^R \mid w \in \{a,b\}^+\}$$

***Also parse the string abba.***

***Or***

***Make a PDA accepting the language of palindromes.***

***(R.G.P.V., Dec. 2005)***

***Sol.*** We use the fact that the symbols are retrieved from a stack in the reverse order of their insertion. When scanning the first part of the string we push consecutive symbol on the stack. For the second part, we compare the current input symbol with the symbol on the top of the stack and keep continuing as long as the two match. Since symbols are retrieved from the stack in reverse of the order in which they were inserted, a complete match will be achieved iff the input is of the form $ww^R$.

The difficulty with this idea is that we do not know the middle of the strings, i.e., where $w$ ends and $w^R$ starts. But the non-deterministic nature of the automaton helps us with this. The PDA correctly guesses where the middle is,

and switches states at that point. The solution to the problem is given as

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

where $Q = \{q_0, q_1, q_2\}$,

$\Sigma = \{a, b\}$,

$\Gamma = \{a, b, Z_0\}$,

$F = \{q_2\}$.

The transition functions can be defined as – A set to push $w$ on the

$$\delta(q_0, a, a) = \{(q_0, aa)\}$$
$$\delta(q_0, b, a) = \{(q_0, ba)\}$$
$$\delta(q_0, a, b) = \{(q_0, ab)\}$$
$$\delta(q_0, b, b) = \{(q_0, bb)\}$$
$$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$$
$$\delta(q_0, b, Z_0) = \{(q_0, bZ_0)\}$$

A set to guess the middle of the strings, where the PDA switches state $q_0$ to $q_1$.

$$\delta(q_0, \lambda, a) = \{(q_1, a)\}$$
$$\delta(q_0, \lambda, b) = \{(q_1, b)\}$$

A set to match $w^R$ against the content of the stack

$$\delta(q_1, a, a) = \{(q_1, \lambda)\}$$
$$\delta(q_1, b, b) = \{(q_1, \lambda)\}$$

and finally, $\delta(q_1, \lambda, Z_0) = \{(q_2, Z_0)\}$

These are the rules to recognize a successful match for $ww^R$.

The sequence of moves in accepting abba is

$$(q_0, abba, Z_0) \vdash (q_0, baa, aZ_0)$$
$$\vdash (q_0, ba, baZ_0)$$
$$\vdash (q_1, ba, baZ_0)$$
$$\vdash (q_1, a, aZ_0)$$
$$\vdash (q_1, \lambda, Z_0)$$
$$\vdash (q_2, \lambda, Z_0)$$

The non-deterministic alternative for locating the middle of the st taken at the third move. At that stage, the PDA is in ID $(q_0, ba, baZ_0)$ two choices for its next move. One is to use $\delta(q_0, b, b) = \{(q_0,$ make the move.

$$(q_0, ba, baZ_0) \vdash (q_0, a, bbaZ_0),$$

the second is the one used above, namely $\delta(q_0, \lambda, b) = \{(q_1, b)\}$; later leads to acceptance of the inputs. Hence,

$$N(A) = \{ww^R | w \in \{a, b\}^+\}$$



***Prob.12. Design PDA for the language –***

$L = \{w | w \in (a + b)^*$ *and* $n_a(w) > n_b(w)\}$. ***(R.G.P.V., Dec. 2011)***

***Or***

***Give transition table for deterministic PDA recognizing the following language –***

$L = \{x \in \{a, b\}^* \mid n_a(x) > n_b(x)\}$. ***(R.G.P.V., Dec. 2005)***

***Or***

***Design a PDA to accept the language*** $\{x \in (a, b)^* | n_a(x) > n_b(x)\}$. ***(R.G.P.V., Dec. 2015)***

***Sol.*** The required PDA $M$ for language

$L = \{w/w \in (a+b)^*$ and $n_a(w) > n_b(w)\}$ is defined as –

$M = (\{q_0, q_f\}, \{a, b\}, \{a, b, Z_0\}, \delta, q_0, Z_0, \{q_f\})$

where $\delta$ is defined as –

$$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$$
$$\delta(q_0, b, Z_0) = \{(q_0, bZ_0)\}$$
$$\delta(q_0, a, a) = \{(q_0, aa)\}$$
$$\delta(q_0, b, b) = \{(q_0, bb)\}$$
$$\delta(q_0, a, b) = \{(q_0, \in)\}$$
$$\delta(q_0, b, a) = \{(q_0, \in)\}$$
$$\delta(q_0, \in, a) = \{(q_f, \in)\}$$

***Prob.13. Define PDA. Construct the PDA accepting*** $\{WCW^R | W$ *in* $(0+1)^*\}$ ***by empty stack. Take input as 001C100.*** ***(R.G.P.V., June 2006)***

***Sol.*** **PDA** – Refer to Q.2.

For a move in which the PDA writes a symbol on the top of the stack, $\delta$ has a value $(q, \gamma)$ where $|\gamma| = 2$. For example, $\delta(q_1, 0, R) = \{(q_1, BR)\}$. If $\gamma$ were of length one, the PDA would simply replace the top symbol by a new symbol and not increase the length of the stack. This allows us to let $\gamma$ equal $\in$ when we wish to pop the stack.

$M = (\{q_1, q_2\}, \{0, 1, C\}, \{R, B, G\}, \delta, q_1, R, \phi)$

$\delta(q_1, 0, R) = \{(q_1, BR)\}$ $\quad$ $\delta(q_1, 1, R) = \{(q_1, GR)\}$

$\delta(q_1, 0, B) = \{(q_1, BB)\}$ $\quad$ $\delta(q_1, 1, B) = \{(q_1, GB)\}$

$\delta(q_1, 0, G) = \{(q_1, BG)\}$ $\quad$ $\delta(q_1, 1, G) = \{(q_1, GG)\}$

$\delta(q_1, C, R) = \{(q_2, R)\}$

$\delta(q_1, C, B) = \{(q_2, B)\}$

$\delta(q_1, C, G) = \{(q_2, G)\}$

$\delta(q_2, 0, B) = \{(q_2, \in)\}$ $\quad$ $\delta(q_2, 1, G) = \{(q_2, \in)\}$

$\delta(q_2, \in, R) = \{(q_2, \in)\}$

***Fig. 4.4 Formal Pushdown Automaton Accepting*** $\{WCW^R | W$ *in* $(0+1)^*\}$ ***by Empty Stack***

Note that the rule $\delta(q_2, \in, R) = \{q_2, \in)\}$ means that the PDA, in state with $R$ the top stack symbol, can erase the $R$ independently of the input symbol. In this case, the input head is not advanced, and in fact, there need not be remaining input. Following fig. 4.4, gives a formal pushdown automaton accepts $\{WCW^R | W \text{ in } (0 + 1)^*\}$ by empty stack.

Following fig. 4.5, shows the accessible IDs of $M$ when $M$ processes string 001$C$100.

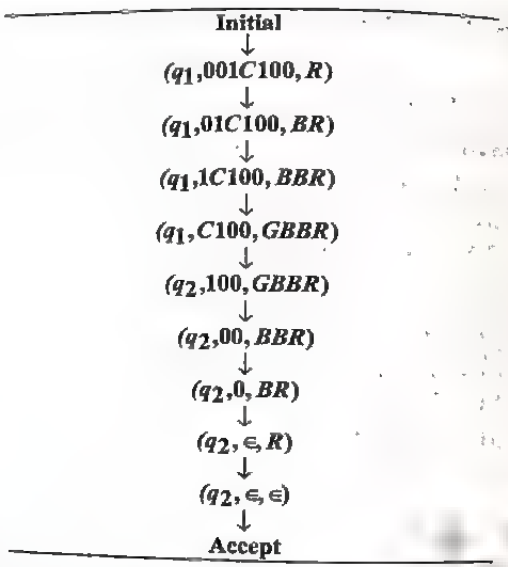



***Fig. 4.5 Accessible IDs for the PDA of Fig. 4.4 with Input 001C10***

***Prob.14. Design a pushdown automata which accepts set of balanced parantheses*** $\{\{[(\,)]\}\}$.

***(R.G.P.V., June 20***

***Sol.*** The pushdown automata is given by –

$$A = (\{q_0\}, \{(, \{, \}, )\}, \{c, \{\}, \delta, q_0, Z_0, \phi)$$

where $\delta$ is given by –

$$\delta(q_0, \{, Z_0) = (q_0, \{\, Z_0)$$
$$\delta(q_0, (, Z_0) = (q_0, (\, Z_0)$$
$$\delta(q_0, \{, \{\,) = (q_0, \{\{\,)$$
$$\delta(q_0, (, (\,) = (q_0, ((\,)$$
$$\delta(q_0, \{, (\,) = (q_0, \{(\,)$$
$$\delta(q_0, (, \{\,) = (q_0, (\{\,)$$
$$\delta(q_0, ), (\,) = (q_0, \in)$$
$$\delta(q_0, \}, \{\,) = (q_0, \in)$$
$$\delta(q_0, \in, Z_0) = (q_0, \in)$$



***Prob.15. Construct PDA for the following set –***

$$L = \{0^n 1^n 2^n | n \geq 1\}$$

***(R.G.P.V., June 2010)***

***Sol.*** Refer to Prob.48 (Unit-III) (assume 0 as a, 1 as b and 2 as c).

Since, the given language is not context-free. So, this language is beyond the recognization power of PDA. Hence, there is no PDA which can determine this language.

***Prob.16. Construct NPDA's that accept the language***

$$L = \{a^n b^m : n \leq m \leq 3n\}$$
$$\text{on } \Sigma = \{a, b\}$$

***Sol.*** The required NPDA can be defined as follows –

$$M = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, b, Z_0\}, \delta, q_0, Z_0, \phi)$$

and $\delta$ is defined by –

$$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$$
$$\delta(q_0, a, a) = \{(q_0, aa)\}$$
$$\delta(q_0, b, a) = \{(q_1, a), (q_0, \in)\}$$
$$\delta(q_1, b, a) = \{(q_2, a), (q_0, \in)\}$$
$$\delta(q_2, b, a) = \{(q_0, \in)\}$$
$$\delta(q_0, \in, Z_0) = \{(q_0, \in)\}$$

## CONVERSION OF PDA TO CFG, CONVERSION OF CFG TO PDA

***Q.15. Prove that, if L = L(M) for some PDA M, then L is a context-free language.***

***Ans.*** **Proof** – Assume that $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \{q_f\})$ satisfies the following conditions –

(i) It has a single final state $q_f$ that is entered iff the stack is empty.

(ii) All transitions must have the form $\delta(q_i, a, A) = \{c_1, c_2, \ldots, c_n\}$, where,

or

$$c_i = (q_i, \lambda)$$
$$c_i = (q_j, BC).$$

i.e., each move either increases or decreases the stack content by a single symbol.

We use the suggested construction to get the grammar $G = (V, T, S, P)$, with $T = \Sigma$ and $V$ consisting of elements of the form $(q_i\, cq_j)$. We will show that the grammar so obtained is such that for all $q_i, q_j \in Q, A \in \Gamma, X \in \Gamma^*, u, v \in \Sigma^*$,

$$(q_i, uv, AX) \vdash^* (q_j, X) \quad \ldots(i)$$

implies that

$$(q_i A q_j) \overset{*}{\Rightarrow} u, \quad \text{and vice versa.}$$

The first part is to prove that, whenever the PDA is such that the sy... $A$ and its effects can be removed from the stack while scanning $u$ and ... from state $q_i$ to $q_j$, then the variable $(q_i A q_j)$ can derive $u$. This is not ... observe since the grammar was explicitly constructed to do this. We ... need an induction on the number of moves to make it precise.

For the converse, consider a single step in the derivation as

$$(q_i A q_k) \Rightarrow a(q_j B q_1)(q_1 C q_k).$$

Using the corresponding transition for the PDA

$$\delta(q_i, a, A) = \{(q_j, BC, \ldots\ldots)\},$$

we observe that $A$ can be removed from the stack, $BC$ put on, reading $a$ ... the control unit going from state $q_i$ to $q_j$. In the same way, if

$$(q_i A q_j) \Rightarrow a,$$

then there must be a corresponding transition

$$\delta(q_i, a, A) = \{(q_j, \lambda)\}$$

whereby the $A$ can be popped off the stack. We observe from this that ... sentential forms derived from $(q_i A q_j)$ define a sequence of poss... configurations of the PDA by which equation (i) can be achieved.

Notice that $(q_i A q_j) \Rightarrow a(q_j B q_1)(q_1 C q_k)$ might be possible for ... $(q_j B q_1)(q_1 C q_k)$ for which there is no corresponding transition of the for... or (iv). But, in that case, at least one of the variables on the right will be use... For all sentential forms leading to a terminal string the argument given ...

If we now use the conclusion to

$$(q_0, w, Z_0) \vdash^* (q_f, \lambda, \lambda),$$

we see that this can be so if and only if

$$(q_0 Z_0 q_f) \overset{*}{\Rightarrow} w$$

Consequently $L(M) = L(G)$

***Q.16. If $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is a PDA, then construct a ... free grammar such that***

$$L(G) = N(A).$$

***Or***

***Show the procedure of converting a PDA into equivalent CFG ... suitable example.*** ***(R.G.P.V., June ...***

***Ans.*** **Construction of $G$** – We define $G = (V, T, P, S)$,

where $V = \{S\} \cup \{[q, Z, q'] \mid q, q' \in Q, Z \in \Gamma\}$

i.e., any element of $V$ is either the new symbol $S$ working as the start ... for $G$ or an ordered triple whose first and third elements are states ... second element is a pushdown symbol.



The productions in $P$ are induced by moves of PDA as follows –

$R_1$ – $S$-productions are given by $S \to [q_0, Z_0, q]$ for every $q$ in $Q$.

$R_2$ – Each move erasing a pushdown symbol given by $(q', \lambda) \in \delta (q, a, Z)$ induce the prodcution $[q, Z, q'] \to a$.

$R_3$ – Each move not erasing a pushdown symbol given by $(q_1, Z_1 Z_2 \ldots Z_m) \in \delta (q, a, Z)$ induces many productions of the form

$$[q, Z, q'] \to a\,[q_1, Z_1, q_2]\,[q_2, Z_2, q_3] \ldots [q_m, Z_m, q']$$

where each of the states $q', q_2 \ldots q_m$ can be any state in $Q$. Each move yields many productions because of $R_3$. It is better understood by taking an example.

Construct a context-free grammar $G$ which accepts $N(A)$, where

$$A = (\{q_0, q_1\}, \{a, b\}, \{Z_0, Z\}, \delta, q_0, Z_0\ \phi)$$

where $\delta$ is given by

$$\delta(q_0, b, Z_0) = \{(q_0, ZZ_0)\}$$
$$\delta(q_0, \lambda, Z_0) = \{(q_0, \lambda)\}$$
$$\delta(q_0, b, Z) = \{(q_0, ZZ)\}$$
$$\delta(q_0, a, Z) = \{(q_1, Z)\}$$
$$\delta(q_1, b, Z) = \{(q_1, \lambda)\}$$
$$\delta(q_1, a, Z_0) = \{q_0, Z_0)\}$$

Now, we define $G$ as follows –

$$G = (V, \{a, b\}, P, S)$$

where $V$ consists of $S$, $[q_0, Z_0, q_0]$, $[q_0, Z_0, q_1]$ $[q_0, Z, q_0]$, $[q_0, Z, q_1]$, $[q_1, Z_0, q_0]$, $[q_1, Z_0, q_1]$, $[q_1, Z, q_0]$, $[q_1, Z, q_0]$.

The productions are –

$$P_1 - S \to [q_0, Z_0, q_0]$$
$$P_2 - S \to [q_0, Z_0, q_1]$$

$\delta(q_0, b, Z_0) = \{(q_0, ZZ_0)\}$ yields

$$P_3 - [q_0, Z_0, q_0] \to b[q_0, Z, q_0]\,[q_0, Z_0, q_0]$$
$$P_4 - [q_0, Z_0, q_0] \to b[q_0, Z, q_1]\,[q_1, Z_0, q_0]$$
$$P_5 - [q_0, Z_0, q_1] \to b[q_0, Z, q_0]\,[q_0, Z_0, q_1]$$
$$P_6 - [q_0, Z_0, q_1] \to b[q_0, Z, q_1]\,[q_1, Z_0, q_1]$$

$\delta(q_0, \lambda, Z_0) = \{q_0, \lambda)\}$ gives

$$P_7 - [q_0, Z_0, q_0] \to \lambda$$

$\delta(q_0, b, Z) = \{(q_0, ZZ)\}$ gives

$$P_8 - [q_0, Z, q_0] \to b[q_0, Z, q_0]\,[q_0, Z, q_0]$$
$$P_9 - [q_0, Z, q_0] \to b[q_0, Z, q_1]\,[q_1, Z, q_0]$$
$$P_{10} - [q_0, Z, q_1] \to b[q_0, Z, q_0]\,[q_0, Z, q_1]$$

$P_{11}$ – $[q_0, Z, q_1] \rightarrow b[q_0, Z, q_1]\,[q_1, Z, q_1]$

$\delta(q_0, a, Z) = \{(q_1, Z)\}$ yields

$P_{12}$ – $[q_0, Z, q_0] \rightarrow a[q_1, Z, q_0]$

$P_{13}$ – $[q_0, Z, q_1] \rightarrow a[q_1, Z, q_1]$

$\delta(q, b, Z) = \{(q_1, \lambda)\}$ gives

$P_{14}$ – $[q_1, Z, q_1] \rightarrow b$

$\delta(q_1, a, Z_0) = \{(q_0, Z_0)\}$ gives

$P_{15}$ – $[q_1, Z_0, q_0] \rightarrow a\,[q_0, Z_0, q_0]$

$P_{16}$ – $[q_1, Z_0, q_1] \rightarrow a\,[q_0, Z_0, q_1]$

Thus, $P_1 - P_{16}$ give the productions in $P$.

***Q.17. Prove that, for any context-free language L, there exists a PDA M such that –***

$$L = L(M)$$

***Ans.* Proof** – If $L$ is a $\lambda$-free context-free language, there exists a context-free grammar in Greibach normal form for it. Let $G = (V, T, P, S)$ be such a grammar. We then construct a PDA which simulates leftmost derivations in this grammar. The simulation will be done in such a way, that the unprocessed part of the sentential form is in the stack, while the terminal prefix of a sentential form matches the corresponding prefix of the input string.

Specifically, the PDA will be

$$M = (\{q_0, q_1, q_f\}, \Sigma, V \cup \{Z_0\}, \delta, q_0, Z_0, \{q_f\})$$

where $\Sigma = T$ and $Z_0 \notin V$. Note that the input alphabets of $M$ are identical with the set of terminals of $G$ and that the stack alphabets contains the set of variables of the grammar.

The transition function will have

$$\delta(q_0, \lambda, Z_0) = \{(q_1, SZ_0)\}$$

so that after the first move of $M$, the stack contains the start symbol $S$ of a derivation. (The stack start symbol $Z_0$ is a marker to allow us to detect the end of the derivation.) In addition, the set of transition rules is such that

$$(q_1, u) \in \delta(q_1, a, A),$$

whenever

$$A \rightarrow au$$

is in $P$. This reads input $a$ and removes the variable $A$ from the stack, replacing it with $u$. In this way, it generates the transitions that allow the PDA to simulate all derivations. Finally, we have

$$\delta(q_1, \lambda, Z_0) = \{(q_f, Z_0)\},$$

to get $M$ into a final state.



To show that $M$ accepts any $w \in L(G)$, consider the partial leftmost derivation as follows –

$$S \overset{*}{\Rightarrow} a_1a_2 \ldots a_nA_1A_2 \ldots A_m$$

$$a_1a_2 \ldots a_nbB_1 \ldots B_k A_2 \ldots A_m.$$

$\Rightarrow$

If $M$ is to simulate this derivation, then after scanning $a_1a_2 \ldots a_n$ the stack must contain $A_1A_2 \ldots A_m$. To take the next step in the derivation, $G$ must have a production as follows –

$$A_1 \rightarrow bB_1 \ldots B_k$$

But the construction is such that then $M$ has a transition rule in which

$$(q_1, B_1 \ldots B_k) \in \delta(q_1, b, A_1)$$

Thus the stack now contains $B_1 \ldots B_k A_2 \ldots A_m$ after having read $a_1a_2 \ldots a_nb$.

A simple induction principle on the number of steps in the derivation then shows that if

$$S \overset{*}{\Rightarrow} w,$$

then

$$(q_1, w, SZ_0) \overset{*}{\vdash} (q_1, \lambda, Z_0)$$

Using rules (i) and (iii) we have

$$(q_0, w, Z_0) \vdash (q_1, w, SZ_0) \overset{*}{\vdash} (q_1, \lambda, Z_0) \vdash (q_f, \lambda, Z_0)$$

so that

$$L(G) \subseteq L(M).$$

To prove that $L(M) \subseteq L(G)$, let $w \in L(M)$. Then by definition

$$(q_0, w, Z_0) \overset{*}{\vdash} (q_f, \lambda, u)$$

But there is only one way to get from $q_0$ to $q_1$ and only one way from $q_1$ to $q_f$. Therefore, we must have the following –

i.e.,

$$(q_1, w, SZ_0) \overset{*}{\vdash} (q_1, \lambda, Z_0)$$

Now let us write $w = a_1a_2a_3 \ldots a_n$. Then the first step in

$$(q_1, a_1a_2a_3 \ldots a_n, SZ_0) \overset{*}{\vdash} (q_1, \lambda, Z_0) \qquad \ldots(iv)$$

must be a rule of the form (ii) to get

$$(q_1, a_1a_2a_3 \ldots a_n, SZ_0) \vdash (q_1, a_2a_3 \ldots a_n, u_1Z_0).$$

But then the grammar has a rule of the form $S \rightarrow a_1u_1$, so that

$$S \Rightarrow a_1u_1.$$

Repeating this, writing $u_1 = Au_2$, we have the following –

$$(q_1, a_2a_3 \ldots a_n, Au_2Z_0) \vdash (q_1, a_3 \ldots a_n, u_3u_2Z_0),$$

It implies that $A \to a_2u_3$ is in the grammar and that

$$S \overset{*}{\Rightarrow} a_1a_2u_3u_2.$$

This makes it quite clear at any point the stack contents (excluding $z_0$) are identical with the unmatched part of the sentential form, so that (iv) is

$$S \overset{*}{\Rightarrow} a_1a_2....a_n.$$

In consequence, $L(M) \subseteq L(G)$, completing the proof if the language does not contain $\lambda$.

If $\lambda \in L$, we add to the constructed PDA the transition –

$$\delta(q_0, \lambda, Z_0) = \{(q_f, Z_0)\}$$

so that the empty string is also accepted.

***Q.18. Prove that the class of language accepted by PDA is exactly the class of CFG, hence construct PDA for the following grammar –***

$$S \to aSa|bSb|a|b|\in$$

***What language is represented by it ?*** ***(R.G.P.V., Dec. 20***

***Ans.*** **Proof** – Assume that $M = (Q, \Sigma, \Gamma, \delta, q_0, z, \{q_f\})$. We use the suggested construction to get the grammar $G = (V, T, S, P)$, with $T = \Sigma$ and consisting of elements of the form $(q_iCq_j)$. We will show that the grammar obtained is such that for all $q_i, q_j \in Q$, $A \in \Gamma$, $X \in \Gamma^*$, $u, v \in \Sigma^*$,

$$(q_i, uv, AX) \overset{*}{\vdash} (q_j, X)$$

implies that $(q_iAq_j) \overset{*}{\Rightarrow} u$,

and vice versa.

The first part is to show that, whenever the NPDA is such that the symbol $A$ and its effects can be removed from the stack while reading $u$ and going from state $q_i$ to $q_j$, then the variable $(q_iAq_j)$ can derive $u$. This is not hard to see since the grammar was explicitly constructed to do this. We only need induction on the number of moves to make this precise.

For the converse, consider a single step in the derivation such as

$$(q_i Aq_k) \Rightarrow a(q_j Bq_l)(q_l Cq_k))$$

Using the corresponding transition for the npda

$$\delta(q_i, a, A) = \{(q_j, BC), ....)\}$$

We see that the $A$ can be removed from the stack, $BC$ put on, reading $a$ with the control unit going from state $q_i$ to $q_j$. Similarly, if

$$(q_i Aq_j) \Rightarrow a$$

Then there must be a corresponding transition

$$\delta(q_i, a, A) = \{(q_j, \lambda)\}$$



whereby the $A$ can be popped off the stack. We see from this that the sentential forms derived from $(q_i Aq_j)$ define a sequence of possible configurations of the NPDA by which equation (i) can be achieved.

Note that $(q_i Aq_j) \Rightarrow a\,(q_j Bq_l)\,(q_l Cq_k)$ might be possible for some $(q_j Bq_l)(q_l Cq_k)$ for which there is no corresponding transition of the form equation (ii) or equation (iii). But, in that case, at least one of the variables on the right will be useless. For all sentential forms leading to a terminal string the argument given holds

If we now apply the conclusion to

$$(q_0, w, Z) \overset{*}{\vdash} (q_f, \lambda, \lambda)$$

This can be so if and only if

$$(q_0zq_f) \overset{*}{\Rightarrow} w$$

Therefore $L(M) = L(G)$

**PDA** – The grammar is not in GNF, therefore, first we obtain an equivalent grammar in GNF, which is

$$S \to aSA|bSB|a|b|\in$$

$$A \to a$$

$$B \to b$$

Therefore, the PDA equivalent to the given grammar is –

$$M = (\{q\}, \{a, b\}, \{S, A, B\}, \delta, q, S, \phi)$$

where, $\delta(q, a, S) = \{(q, SA), (q, \in)\}$, because of $S \to a\,SA$, and $S \to a$

$\delta(q, b, S) = \{(q, SB), (q, \in)\}$ because of $S \to bSB$ and $S \to b$

$\delta(q, a, A) = \{(q, \in)\}$, because of $A \to a$

$\delta(q, b, B) = \{(q, \in)\}$, because of $B \to b$

**Language** – $L(G) = \{w : w \in$ set of all palindromes over $\{a, b\}\}$

***Q.19. If L is a context-free language, then construct a PDA A accepting L by empty store, i.e., L = N(A).***

***Or***

***Write short note on equivalence of CFGs for PDA.***

***(R.G.P.V., June 2005)***

***Ans.*** We construct the PDA $A$ by making use of productions in $G$.

**Construction of $A$** – Let $L = L(G)$, where $G = (V, T, P, S)$ is a context-free grammar. We construct a PDA $A$ as follows –

$$A = (\{q\}, \Sigma, V \cup T, \delta, q, S, \phi)$$

where $\delta$ is defined by the following rules –

$$R_1 - \delta(q, \lambda, A) = \{(q, \alpha) \mid A \to \alpha \text{ is in } P\}$$

$$R_2 - \delta(q, a, a) = \{(q, \lambda\} \text{ for every } a \text{ in } \Sigma$$

The construction can be explained as follows –

The pushdown symbols in $A$ are variables and terminals. If the PDA reads a variable $A$ on the top of the stack, it makes a $\lambda$-move by placing the R.H.S. of any $A$-production (after erasing $A$). If the PDA reads a terminal $a$ on the top of the stack and if it matches with the current input symbol, then the PDA erases $a$. The PDA halts in other cases.

If $w \in L(G)$ is obtained by a leftmost derivation

$$S \Rightarrow u_1A_1\alpha_1 \Rightarrow u_1u_2A_2\alpha_2\alpha_1 \Rightarrow \ldots \Rightarrow w,$$

then $A$ can empty the stack on application of input string $w$. The first move $a$ is by a $\lambda$-move corresponding to $S \to u_1A_1\alpha_1$. The PDA erases $S$ and stores $u_1A_1\alpha_1$. Then using $R_2$, the PDA erases the symbols in $u_1$ by processing prefix of $w$. Now, the topmost symbol in the stack is $A_1$. Once again applying the $\lambda$-move corresponding to $A_1 \to u_2A_2\alpha_2$, the PDA erases $A_1$ stores $u_2A_2\alpha_2$ above $\alpha_1$. Proceeding in this way, the PDA empties the by processing the entire string $w$. Let us take an example.

Construct a PDA $A$ equivalent to the following CFG – $S \to 0BB, B \to$ $\mid 1S \mid 0$. Test whether $010^4$ is in $N(A)$.

Now, we define PDA $A$ as follows –

$$A = (\{q\}, \{0, 1\}, \{S, B, 0, 1\}, \delta, q, S, \phi)$$

where $\delta$ is defined by the following rules –

$$R_1 - \delta(q, \lambda, S) = \{(q, 0BB)\}$$

$$R_2 - \delta(q, \lambda, B) = \{(q, 0S), (q, 1S), (q, 0)\}$$

$$R_3 - \delta(q, 0, 0) = \{(q, \lambda)\}$$

$$R_4 - \delta(q, 1, 1) = \{(q, \lambda)\}$$

Let us test whether $010^4$ is in $N(A)$ or not.

$(q, 010^4, S) \vdash (q, 010^4, 0BB)$ by rule $R_1$

$\vdash (q, 10^4, BB)$ by rule $R_3$

$\vdash (q, 10^4, 1SB)$ by rule $R_2$, since $(q, 1S) \in \delta(q, \lambda)$

$\vdash (q, 0^4, SB)$ by rule $R_4$

$\vdash (q, 0^4, 0BBB)$ bu rule $R_1$

$\vdash (q, 0^3, BBB)$ by rule $R_2$, since $(q, 0) \in \delta(q, \lambda)$

$\vdash^* (q, \lambda, \lambda)$ by rule $R_3$

Thus, $010^4 \in N(A)$



Note that after entering $(q, 10^4, BB)$, the PDA may halt for a different sequence of moves, for example, $(q, 10^4, BB) \vdash (q, 10^4, 0B) \vdash (q, 10^4, 00)$. As $\delta(q, 1, 0)$ is the empty set, the PDA halts.

## NUMERICAL PROBLEMS

***Prob.17. Consider a PDA A such that –***

$$A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

$$Q = \{q_0, q_1\}, \Sigma = \{a, b\}, \Gamma = (a, Z_0), F = \phi,$$

*where*

*and $\delta$ is given by*

$$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$$

$$\delta(q_0, a, a) = \{(q_0, aa)\}$$

$$\delta(q_0, b, a) = \{(q_1, \lambda)\}$$

$$\delta(q_1, b, a) = \{(q_1, \lambda)\}$$

$$\delta(q_1, \lambda, Z_0) = \{(q_1, \lambda)\}$$

***Sol.*** The given PDA is defined as follows –

$A = (\{q_0, q_1\}, \{a, b\}, \{a, Z_0\}, \delta, q_0, Z_0, \phi)$, where $\delta$ is given by –

$$R_1 - \delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$$

$$R_2 - \delta(q_0, a, a) = \{(q_0, aa)\}$$

$$R_3 - \delta(q_0, b, a) = \{(q_1, \lambda)\}$$

$$R_4 - \delta(q_1, b, a) = \{(q_1, \lambda)\}$$

$$R_5 - (q_1, \lambda, Z_0) = \{(q_1, \lambda)\}$$

$R_1$ is used to store $a$ in the stack if it is the first symbol of an input string. $R_2$ can be used repeatedly to store $a^n$ in the stack. When $b$ is encountered for the first time in the input string, $a$ is erased (in the stack) using $R_3$. Also, the PDA makes a transition to state $q_1$. After processing the entire input string, if $Z_0$ remains in the stack, it can be erased using the null move given by $R_5$. So, if $u = a^nb^n$, then we have

$$(q_0, a^nb^n, Z_0) \vdash^* (q_0, b^n, a^nZ_0) \quad \text{by applying } R_1 \text{ and } R_2$$

$$\vdash^* (q_1, \lambda, Z_0) \quad \text{by applying } R_3 \text{ and } R_4$$

$$\vdash (q_1, \lambda, \lambda) \quad \text{by applying } R_5$$

Therefore, $a^nb^n \in N(A)$.

If $u \in N(A)$, then $(q_0, u, Z_0) \vdash^* (q_1, \lambda, \lambda)$ (Note that the stack can be empty only when $A$ is in state $q_1$). Also, $u$ should start with $a$. Otherwise, we cannot make any move. We store the symbol $a$ in the stack if the current input

symbol is $a$ and the topmost symbol in the stack is $a$ or $Z_0$. On looking input symbol $b$, the PDA erases the symbol $a$ in the stack. The PDA enters ID $(q_1, \lambda, \lambda)$only by the application of $R_5$. The PDA can reach the ID ($Z_0$) only by erasing the $a$'s in the stack. This is possible only when the of $b$'s is equal to number of $a$'s, and so $u = a^n b^n$. Thus, we have prove

$$N(A) = \{a^n b^n | n \geq 1\}.$$

Now, let $B = (Q', \{a,b\}, \Gamma', \delta_B, q'_0, Z'_0, F)$

where $Q' = \{q_0, q'_0, q_1, q_f\}$,

$F' = \{q_f\}, \Gamma' - \{a, b, Z'_0\}$

and $\delta_B$ is defined by –

$$\delta_B(q'_0, \lambda, Z'_0) = \{(q_0, Z_0 Z'_0)\}$$
$$\delta_B(q_0, a, Z_0) = \{(q_0, aZ_0)\}$$
$$\delta_B(q_0, a, a) = \{(q_0, aa)\}$$
$$\delta_B(q_0, b, a) = \{(q_1, \lambda)\}$$
$$\delta_B(q_1, b, a) = \{(q_1, \lambda)\}$$
$$\delta_B(q_1, \lambda, Z_0) = \{(q_1, \lambda)\}$$
$$\delta_B(q_0, \lambda, Z'_0) = \{(q_f, \lambda)\}$$
$$\delta_B(q_1, \lambda, Z'_0) = \{(q_f, \lambda)\}$$

Thus, $L(B) = N(A) = \{a^n b^n | n \geq 1\}$

***Prob.18. Give a grammar for the language N(M) where –***

$$M = (\{q_0, q_1\}, \{0, 1\}, \{x, Z_0\}, \delta, q_0, Z_0)$$

***where δ is given by –***

$$\delta(q_0, 0, Z_0) = \{(q_0, x Z_0)\}$$
$$\delta(q_0, 0, x) = \{(q_0, x x)\}$$
$$\delta(q_0, 1, x) = \{(q_1, \in)\}$$
$$\delta(q_1, 1, x) = \{(q_1, \in)\}$$
$$\delta(q_1, \in, x) = \{(q_1, \in)\}$$
$$\delta(q_1, \in, Z_0) = \{(q_1, \in)\}$$

***(R.G.P.V., Dec.***

***Or***

***Construct CFG for the following PDA –***

$$M = (\{q_0, q_1\}, \{0, 1\}, \{Z_0, X\}, \delta, q_0, Z_0)$$



$$\delta(q_0, 0, Z_0) = (q_0, X Z_0)$$
$$\delta(q_1, \in, X) = (q_1, \in)$$
$$\delta(q_0, 1, X) = (q_1, \in)$$
$$\delta(q_0, 0, X) = (q_0, XX)$$
$$\delta(q_1, 1, X) = (q_1, \in)$$
$$\delta(q_1, \in, Z_0) = (q_1, \in)$$

***(R.G.P.V., June 2007)***

***Sol.*** Let $G = (V, \{0, 1\}, P, S)$

where $V$ consists of $S$, $[q_0, Z_0, q_0]$, $[q_0, Z_0, q_1]$, $[q_0, x, q_0]$ $[q_0, x, q_1]$ $[q_1, Z_0, q_0]$, $[q_1, Z_0, q_1]$, $[q_1, x, q_0]$, $[q_1, x, q_1]$.

The productions are –

$P_1 - S \rightarrow [q_0, Z_0, q_0]$

$P_2 - S \rightarrow [q_0, Z_0, q_1]$

$\delta(q_0, 0, Z_0) = \{(q_0, xZ_0)\}$ yields

$P_3 - [q_0, Z_0, q_0] \rightarrow 0[q_0, x, q_0]\ [q_0, Z_0, q_0]$

$P_4 - [q_0, Z_0, q_0] \rightarrow 0[q_0, x, q_1]\ [q_1, Z_0, q_0]$

$P_5 - [q_0, Z_0, q_1] \rightarrow 0[q_0, x, q_0]\ [q_0, Z_0, q_1]$

$P_6 - [q_0, Z_0, q_1] \rightarrow 0\ [q_0, x, q_1]\ [q_1, Z_0, q_1]$

$\delta(q_0, 0, x) = \{(q_0, xx)\}$ gives

$P_7 - [q_0, x, q_0] \rightarrow 0\ [q_0, x, q_0]\ [q_0, x, q_0]$

$P_8 - [q_0, x, q_0] \rightarrow 0\ [q_0, x, q_1]\ [q_1, x, q_0]$

$P_9 - [q_0, x, q_1] \rightarrow 0\ [q_0, x, q_0]\ [q_0, x, q_1]$

$P_{10} - [q_0, x, q_1] \rightarrow 0[q_0, x, q_1]\ [q_1, x, q_1]$

$\delta(q_0, 1, x) = \{(q_1, \lambda)\}$ gives

$P_{11} - [q_0, x, q_1] \rightarrow 1$

$\delta(q_1, 1, x) = \{(q_1, \lambda)\}$ gives

$P_{12} - [q_1, x, q_1] \rightarrow 1$

$\delta(q_1, \lambda, x) = \{(q_1, \lambda)\}$ gives

$P_{13} - [q_1, x, q_1] \rightarrow \lambda$

$\delta(q_1, \lambda, Z_0) = \{(q_1, \lambda)\}$ gives

$P_{14} - [q_1, Z_0, q_1] \rightarrow \lambda$

$P_1 - P_{14}$ give the productions in $P$. **Hence, $G$ is the required grammar.**

***Prob.19. Find a context-free grammar that generates the language accepted by the NPDA –***

$$M = (\{q_0, q_1\}, \{a, b\}, \{A, Z\}, \delta, q_0, Z, \{q_1\})$$

***with transitions –***

$$\delta(q_0, a, Z) = \{(q_0, AZ)\}$$
$$\delta(q_0, b, A) = \{(q_0, AA)\}$$
$$\delta(q_0, a, A) = \{(q_1, \lambda)\}$$

***Or***

***Obtain CFG for the PDA given as below –***

$$A = (\{q_0, q_1\}, \{0, 1\}, \{A, Z\}, \delta, Z, \{q_1\})$$

***where $\delta$ is as given below –***

$$\delta(q_0, 0, Z) = (q_0, AZ)$$
$$\delta(q_0, 1, A) = (q_0, AA)$$
$$\delta(q_0, 0, A) - (q_1, \in)$$

***(R.G.P.V., June 20...***

***Sol.*** Let, $G = (V_N, \{a, b\}, P, S)$

where $V_N$ consists of $S$, $[q_0, A, q_0]$, $[q_0, A, q_1]$, $[q_0, Z, q_0]$, $[q_0, Z, q_1]$, $[q_1, A, q_0]$, $[q_1, A, q_1]$, $[q_1, Z, q_0]$, $[q_1, Z, q_1]$.

The productions are

$P_1 - S \to [q_0, Z, q_0]$

$P_2 - S \to [q_0, Z, q_1]$

$\delta(q_0, a, Z) \to \{(q_0, AZ)\}$ yields

$P_3 - [q_0, Z, q_0] = a\,[q_0, A, q_0]\,[q_0, Z, q_0]$

$P_4 - [q_0, Z, q_0] \to a\,[q_0, A, q_1]\,[q_1, Z, q_0]$

$P_5 - [q_0, Z, q_1] \to a\,[q_0, A, q_0]\,[q_0, Z, q_1]$

$P_6 - [q_0, Z, q_1] \to a\,[q_0, A, q_1]\,[q_1, Z, q_1]$

$\delta(q_0, b, A) \to \{(q_0, AA)\}$ gives

$P_7 - [q_0, A, q_0] \to b\,[q_0, A, q_0]\,[q_0, A, q_0]$

$P_8 - [q_0, A, q_0] \to b\,[q_0, A, q_1]\,[q_1, A, q_0]$

$P_9 - [q_0, A, q_1] \to b\,[q_0, A, q_0]\,[q_0, A, q_1]$

$P_{10} - [q_0, A, q_0] \to b\,[q_0, A, q_1]\,[q_1, A, q_1]$

$\delta(q_0, a, A) \to \{(q_1, \lambda)\}$ gives

$P_{11} - [q_0, A, q_1] \to a$

$P_{12} - [q_0, A, q_0] \to a$

$P_1 - P_{12}$ give the productions in $P$.



***Prob.20. Construct a PDA accepting $\{a^n b^m a^n | m, n \geq 1\}$ by null store. Construct the corresponding context-free grammar accpeting the same set.***

***Or***

***Design DPDA for $L = \{0^n 1^m 0^n | m, n \geq 1\}$.*** ***(R.G.P.V., June 2011)***

***Or***

***Construct a PDA accepting $\{a^n b^m a^n \mid m, n \geq 1\}$ by empty store.*** ***(R.G.P.V., June 2006)***

***Or***

***Design PDA to accept the language $L(G) = \{a^n b^m a^n / m, n \geq 1\}$.*** ***(R.G.P.V., Dec. 2016)***

***Sol.*** The PDA $A$ accepting $\{a^n b^m a^n | m, n \geq 1\}$ is defined as follows –

$$A = (\{q_0, q_1\}, \{a, b\}, \{a, Z_0\}, \delta, q_0, Z_0, \phi)$$

where $\delta$ is given by

$R_1 - \delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$

$R_2 - \delta(q_0, a, a) = \{(q_0, aa)\}$

$R_3 - \delta(q_0, b, a) = \{(q_1, a)\}$

$R_4 - \delta(q_1, b, a) = \{(q_1, a)\}$

$R_5 - \delta(q_1, a, a) = \{(q_1, \lambda)\}$

$R_6 - \delta(q_1, \lambda, Z_0) = \{(q_1, \lambda)\}$

We start storing $a$'s until one $b$ occurs (rules $R_1$ and $R_2$). When the current input symbol is $b$, the state changes, but no change in the stack occurs (rule $R_3$). Once all the $b$'s in the input string are exhausted (using rule $R_4$), the remaining $a$'s are erased (rule $R_5$). $Z_0$ is erased using rule $R_6$. So

$$(q_0, a^n b^m a^n, Z_0) \vdash^* (q_1, \lambda, Z_0) \vdash (q_1, \lambda, \lambda)$$

This means that $a^n b^m a^n \in N(A)$. We can show that

$$N(A) = \{a^n b^m a^n | m, n \geq 1\}$$

by using rules $R_1 - R_6$.

Now, we define the grammar $G = (V, \{a, b\}, P, S)$, where $V$ consists of $S$, $[q_0, Z_0, q_0]$, $[q_0, Z_0, q_1]$, $[q_0, a, q_0]$ $[q_0, a, q_1]$, $[q_1, Z_0, q_0]$, $[q_1, Z_0, q_1]$, $[q_1, a, q_0]$, $[q_1, a, q_1]$.

The productions in $P$ are constructed as follows –

$P_1 - S \to [q_0, Z_0, q_0]$

$P_2 - S \to [q_0, Z_0, q_1]$

$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$ induces

$P_3 - [q_0, Z_0, q_0] \to a[q_0, a, q_0]\,[q_0, Z_0, q_0]$

$P_4$ – $[q_0, Z_0, q_0] \rightarrow a\,[q_0, a, q_1]\,[q_1, Z_0, q_0]$

$P_5$ – $[q_0, Z_0, q_1] \rightarrow a\,[q_0, a, q_0]\,[q_0, Z_0, q_1]$

$P_6$ – $[q_0, Z_0, q_1] \rightarrow a[q_0, a, q_1]\,[q_1, Z_0, q_1]$

$\delta(q_0, a, a) = \{(q_0, aa)\}$ yields

$P_7$ – $[q_0, a, q_0] \rightarrow a\,[q_0, a, q_0]\,[q_0, a, q_0]$

$P_8$ – $[q_0, a, q_0] \rightarrow a\,[q_0, a, q_1]\,[q_1, a, q_0]$

$P_9$ – $[q_0, a, q_1] \rightarrow a\,[q_0, a, a_0]\,[q_0, a, q_1]$

$P_{10}$ – $[q_0, a, q_1] \rightarrow a[q_0, a, q_1]\,[q_1, a, q_1]$

$\delta(q_0, b, a) = \{(q_1, a)\}$ yields

$P_{11}$ – $[q_0, a, q_0] \rightarrow b\,[q_1, a, q_0]$

$P_{12}$ – $[q_0, a, q_1] \rightarrow b\,[q_1, a, q_1]$

$\delta(q_1, b, a) = \{(q_1, a)\}$ gives

$P_{13}$ – $[q_1, a, q_0] \rightarrow b[q_1, a, q_0]$

$P_{14}$ – $[q_1, a, q_1] \rightarrow b[q_1, a, q_1]$

$\delta(q_1, a, a) = \{(q_1, \lambda)\}$ gives

$P_{15}$ – $[q_1, a, q_1] \rightarrow a$

$\delta(q_1, \lambda, Z_0) = \{(q_1, \lambda)\}$ yields

$P_{16}$ – $[q_1, Z_0, q_1] \rightarrow \lambda$

$P_1 - P_{16}$ give the productions in $P$. **Hence, $G$ is the required grammar.**

***Prob.21. Construct the CFG corresponding to PDA***

$A = (\{q_0, q_1\}, \{a, b\}, \{a, Z_0\}, \delta, q_0, Z_0, \phi)$ *and* $\delta$ *is given by*

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$

$\delta(q_0, a, a) = (q_0, aa)$

$\delta(q_0, b, a) = (q_1, a)$

$\delta(q_1, b, a) = (q_1, a)$

$\delta(q_1, a, a) = (q_1, \lambda)$

$\delta(q_1, \lambda, Z_0) = (q_1, \lambda)$

***Sol.*** Refer to Prob.20.

***Prob.22. Convert the following PDA into CFG –***

$$M = (\{q_0, q_1\}, \{0, 1\}, \{Z_0, x\}, \delta, q_0, Z_0, \phi)$$

*where* $\delta$ *is defined as follows –*

$\delta(q_0, 1, Z_0) = \{(q_0, xZ_0)\}$

$\delta(q_0, 1, x) = \{(q_0, xx)\}$



$\delta(q_0, 0, x) = \{(q_1, x)\}$

$\delta(q_1, 1, x) = \{(q_1, \in)\}$

$\delta(q_1, 0, Z_0) = \{(q_0, Z_0)\}$

$\delta(q_0, \in, Z_0) = \{(q_0, \in)\}$

***(R.G.P.V., June 2003, Dec. 2003, June 2004, 2009)***

***Or***

***Give CFG equivalent to the following PDA –***

$$M = (\{q_0, q_1\}, \{0, 1\}, \{Z_0, X\}, \delta, q_0, Z_0, \phi)$$

***Here, '$\delta$' is given by***

$\delta(q_0, 1, Z_0) = (q_0, XZ_0)$

$\delta(q_0, \in, Z_0) = (q_0, \in)$

$\delta(q_0, 1, X) = (q_0, XX)$

$\delta(q_1, 1, X) = (q_1, \in)$

$\delta(q_0, 0, X) = (q_1, X)$

$\delta(q_1, 0, Z_0) = (q_0, Z_0)$

***(R.G.P.V., Dec. 2006)***

***Sol.*** Let $G = (V, \{0, 1\}, P, S)$

where $V$ consists of $S$, $[q_0, Z_0, q_0]$, $[q_0, Z_0, q_1]$, $[q_0, x, q_0]$, $[q_0, x, q_1]$, $[q_1, Z_0, q_0]$, $[q_1, Z_0, q_1]$, $[q_1, x, q_0]$, $[q_1, x, q_1]$.

The productions are –

$P_1 - S \rightarrow [q_0, Z_0, q_0]$

$P_2 - S \rightarrow [q_0, Z_0, q_1]$

$\delta(q_0, 1, Z_0) = \{(q_0, xZ_0)\}$ yields

$P_3 - [q_0, Z_0, q_0] \rightarrow 1[q_0, x, q_0]\,[q_0, Z_0, q_0]$

$P_4 - [q_0, Z_0, q_0] \rightarrow 1[q_0, x, q_1]\,[q_1, Z_0, q_0]$

$P_5 - [q_0, Z_0, q_1] \rightarrow 1\,[q_0, x, q_0]\,[q_0, Z_0, q_1]$

$P_6 - [q_0, Z_0, q_1] \rightarrow 1[q_0, x, q_1]\,[q_1, Z_0, q_1]$

$\delta(q_0, 1, x) = \{(q_0, xx)\}$ gives

$P_7 - [q_0, x, q_0] \rightarrow 1[q_0, x, q_0]\,[q_0, x, q_0]$

$P_8 - [q_0, x, q_0] \rightarrow 1[q_0, x, q_1]\,[q_1, x, q_0]$

$P_9 - [q_0, x, q_1] \rightarrow 1[q_0, x, q_0]\,[q_0, x, q_1]$

$P_{10} - [q_0, x, q_1] \rightarrow 1[q_0, x, q_1]\,[q_1, x, q_1]$

$\delta(q_0, 0, x) = \{(q_1, x)\}$ yields

$P_{11} - [q_0, x, q_0] \rightarrow 0[q_1, x, q_0]$

$P_{12} - [q_0, x, q_1] \rightarrow 0[q_1, x, q_1]$

$\delta(q_1, 1, x) = \{(q_1, \in)\}$ gives

$P_{13} - [q_1, x, q_1] \rightarrow 1$

$\delta(q_1, 0, Z_0) = \{(q_0, Z_0)\}$ gives

$P_{14} - [q_1, Z_0, q_0] \rightarrow 0\ [q_0, Z_0, q_0]$

$P_{15} - [q_1, Z_0, q_1] \rightarrow 0[q_0, Z_0, q_1]$

$\delta(q_0, \in, Z_0) = \{(q_0, \in)\}$ gives

$P_{16} - [q_0, Z_0, q_0] \rightarrow \in$

$P_1 - P_{16}$ give the productions in $P$. ***Hence, G is the required CFG.***

***Prob.23. Convert the PDA –***

$P = (\{p, q\}, \{0, 1\}, \{x, Z_0\}, \delta, q, Z_0)$

***to a CFG, if $\delta$ is given by –***

$\delta(q, 1, Z_0) = \{(q, xZ_0)\}$

$\delta(q, 1, x) = \{(q, xx)\}$

$\delta(q, 0, x) = \{(p, x)\}$

$\delta(q, \in, Z_0) = \{(q, \in)\}$

$\delta(p, 1, x) = \{(p, \in)\}$

$\delta(p, 0, Z_0) = \{(q, Z_0)\}$

***(R.G.P.V., Dec. 2008)***

***Sol.*** Refer to Prob.22.

**Note** – Replace $q_0$ by $q$ and $q_1$ by $p$.

***Prob.24. For the given CFG, construct a PDA that accepts the same language they generate –***

$S \rightarrow Saa \mid aSa \mid aaS.$ ***(R.G.P.V., Dec. 2009)***

***Sol.*** The PDA $A = (\{q\}, \{a\}, \{S\}, \delta, q, S, \phi)$

where $\delta$ is defined as follows –

$R_1 - \delta(q, \Lambda, S) = \{(q, Saa), (q, aSa), (aaS)\}$

$R_2 - \delta(q, a, a) = \{(q, \Lambda)\}$

***Prob.25. Design PDA corresponding to given CFG –***

$S \rightarrow aSA,\ S \rightarrow bSb,\ S \rightarrow a$ ***(R.G.P.V., Dec. 2013)***

***Sol.*** The grammar is not in GNF. Hence, first we get an equivalent grammar in GNF, which is –

$S \rightarrow aSA \mid bSB \mid a$

$A \rightarrow a$

$B \rightarrow b$



Hence, the PDA equivalent to the grammar is –

$M = (\{q\}, \{a, b\}, \{S, A, B\}, \delta, q, S, \phi)$

where,

$\delta(q, a, S) = \{(q, SA), (q, \in)\}$, because of $S \rightarrow aSA$, and $S \rightarrow a$

$\delta(q, b, S) = \{q, SB\}$, because of $S \rightarrow bSB$

$\delta(q, a, A) = \{(q, \in)\}$, because of $A \rightarrow a$

$\delta(q, b, B) = \{(q, \in)\}$, because of $B \rightarrow b$

***Prob.26. Design PDA corresponding to given CFG***

$S \rightarrow aSa$

$S \rightarrow bSb$

$S \rightarrow c$

***(R.G.P.V., Nov. 20***

***Sol.*** The grammar is not in GNF. Hence, first we get an equivalent gramm in GNF, which is

$S \rightarrow aSA \mid bSB \mid c$

$A \rightarrow a$

$B \rightarrow b$

Hence, the PDA equivalent to the grammar is

$M = (\{q\}, \{a, b, c\}, \{S, A, B\}, \delta, q, S, \phi)$

where,

$\delta(q, a, S) = \{(q, SA)\}$, because of $S \rightarrow aSA$

$\delta(q, b, S) = \{(q, SB)\}$, because of $S \rightarrow bSB$

$\delta(q, a, A) = \{(q, \in)\}$, because of $A \rightarrow a$

$\delta(q, b, B) = \{(q, \in)\}$, because of $B \rightarrow b$

$\delta(q, c, S) = \{(q, \in)\}$, because of $S \rightarrow c$

***Prob.27. Construct a PDA equivalent to the following context-free grammar –***

$S \rightarrow 0BB$

$B \rightarrow 0S \mid 1S \mid 0$

***Test whether 010000 is in N (PDA).*** ***(R.G.P.V., June 2006)***

***Or***

***Define a PDA ? Construct a PDA equivalent to the following context free grammar $S \rightarrow 0BB$, $B \rightarrow 0S \mid 1S \mid 0$. Test whether $010^4$ is in N(A).*** ***(R.G.P.V., Dec. 2017)***

***Sol.*** **PDA** – Refer to Q.2.

PDA $A$ equivalent to the context free grammar is as follows –

$$A = (\{q\}, \{0, 1\}, \{S, B, 0, 1\}, \delta, q, S, \phi)$$

$\delta$ is defined by the following rules –

$R_1 - \delta(q, \Lambda, S) = \{(q, 0BB)\}$

$R_2 - \delta(q, \Lambda, B) = \{(q, 0S), (q, 1S), (q, 0)\}$

$R_3 - \delta(q, 0, 0) = \{(q, \Lambda)\}$

$R_4 - \delta(q, 1, 1) = \{(q, \Lambda)\}$

$(q, 010000, S) \vdash (q, 010000, 0BB)$ by rule $R_1$

$\vdash (q, 10000, BB)$ by rule $R_2$

$\vdash (q, 10000, 1SB)$ by rule $R_4$

Since $(q, 1S) \in \delta(q, \Lambda, B)$

$\vdash (q, 0000, SB)$ by rule $R_2$

$\vdash (q, 0000, 0BBB)$ by rule $R_1$

$\vdash (q, 000, BBB)$ by rule $R_3$

$\vdash^* (q, 000, 000)$ by rule $R_3$

Since $(q, 0) \in \delta(q, 0, B)$

$\vdash^* (q, \Lambda, \Lambda)$ by rule $R_3$

Thus, $010000 \in N(\text{PDA})$.

***Prob.28. Construct the PDA equivalent to the following grammar –***

$$S \rightarrow aAA$$

$$A \rightarrow aS|bS|a$$

***(R.G.P.V., June 2003, Dec. 2006, June 2009, 2010, Dec. 2012)***

***Sol.*** Refer to Prob.27.

**Note** – Here, replace $a$ by 0, $A$ by $B$ and $b$ by 1.

***Prob.29. Construct PDA for the following CFG –***

$$S \rightarrow aB \mid bA$$

$$A \rightarrow a \mid aS \mid bAA$$

$$B \rightarrow b \mid bS \mid aBB.$$

***(R.G.P.V., June 2007)***

***Or***

***Convert the CFG into PDA –***

$$S \rightarrow aB \mid bA$$

$$A \rightarrow a|aS|bAA$$

$$B \rightarrow b|bS|aBB$$

***(R.G.P.V., Dec. 2005)***



***Sol.*** The required PDA $A$ can be defined as follows –

$$A = (\{q\}, \{a, b\}, \{S, A, B, a, b\}, \delta, q, S, \phi)$$

where, $\delta$ is defined by the following rules –

$R_1 - \delta(q, \Lambda, S) = \{(q, aB), (q, bA)\}$

$R_2 - \delta(q, \Lambda, A) = \{(q, a), (q, aS), (q, bAA)\}$

$R_3 - \delta(q, \Lambda, B) = \{(q, b), (q, bS), (q, aBB)\}$

$R_4 - \delta(q, a, a) = \{(q, \Lambda)\}$

$R_5 - \delta(a, b, b) = \{(q, \Lambda)\}$

The language accepted by grammar is

$$L = \{w \in \{a, b\}^* | \, n_a(w) = n_b(w)\}$$

Let us see the sequence of moves in processing the string *baab*

$(q, baab, S) \vdash (q, baab, bA)$ by rule $R_1$

$\vdash (q, aab, A)$ by rule $R_5$

$\vdash (q, aab, aS)$ by rule $R_2$

$\vdash (q, ab, S)$ by rule $R_4$

$\vdash (q, ab, aB)$ by rule $R_1$

$\vdash (q, b, B)$ by rule $R_4$

$\vdash (q, b, b)$ by rule $R_3$

$\vdash (q, \Lambda, \Lambda)$ by rule $R_5$

Thus, $baab \in N(A)$

**Hence, A is the required PDA.**

***Prob.30. Convert the following C.F.G into PDA –***

$$S \rightarrow aS|aA$$

$$A \rightarrow bA|b$$

***(R.G.P.V., June 2008)***

***Sol.*** The required PDA $A$ can be defined as follows –

$$A = (\{q\}, \{a, b\}, \{S, A, a, b\}, \delta, q, s, \phi)$$

where $\delta$ is defined by the following rules –

$R_1 - \delta(q, \Lambda, S) = \{(q, aS), (q, aA)\}$

$R_2 - \delta(q, \Lambda, A) = \{(q, bA), (q, b)\}$

$R_3 - \delta(q, a, a) = \{q, \Lambda)\}$

$R_4 - \delta(q, b, b) = \{(q, A)\}$

Let us see the sequence of moves in processing the string *aabb*.

$(q, aabb, S) \vdash (q, aabb, aS)$ by rule $R_1$
$\vdash (q, abb, S)$ by rule $R_3$
$\vdash (q, abb, aA)$ by rule $R_1$
$\vdash (q, bb, A)$ by rule $R_3$
$\vdash (q, bb, bA)$ by rule $R_2$
$\vdash (q, b, A)$ by rule $R_4$
$\vdash (q, b, b)$ by rule $R_2$
$\vdash (q, \Lambda, \Lambda)$ by rule $R_4$

Thus $aabb \in N(A)$

**Hence, $A$ is the required PDA.**

***Prob.31. Construct PDA for the following grammar –***

$S \rightarrow AB$
$A \rightarrow CD$
$B \rightarrow b$
$C \rightarrow a$
$D \rightarrow a$

***(R.G.P.V., Dec. 201***

***Sol.*** The grammar is not in GNF, first we obtain an equivalent gramm in GNF, which is

$S \rightarrow aDB$
$A \rightarrow aD$
$B \rightarrow b$
$C \rightarrow a$
$D \rightarrow a$

Therefore, the PDA equivalent to the given grammar is –

$M = (\{q\}, \{a, b\}, \{S, A, B, C, D, a, b\}, \delta, q, S, \phi)$
$R_1 - (q, \Lambda, S) = \{(q, aDB)\}$
$R_2 - (q, \Lambda, A) = \{(q, aD)\}$
$R_3 - (q, \Lambda, B) = \{(q, b)\}$
$R_4 - (q, \Lambda, C) = \{(q, a)\}$
$R_5 - (q, \Lambda, D) = \{(q, a)\}$
$R_6 - (q, a, a) = \{(q, \Lambda)\}$
$R_7 - (q, b, b) = \{(q, \Lambda)\}$



***Prob.32. Consider the grammar***

$S \rightarrow aA$
$A \rightarrow aABC \mid bB \mid a$
$B \rightarrow b$
$C \rightarrow c$

***Construct PDA corresponding to this grammar. Also provide moves of the PDA and the leftmost derivation for any string in the language defined by the grammar.*** ***(R.G.P.V., Dec. 2015)***

***Sol.*** The given grammar is already in Greibach normal form. In addition to rules

$\delta(q_0, \lambda, Z) = \{(q_1, SZ)\}$

and $\delta(q_1, \lambda, Z) = \{(q_f, Z)\}$

the PDA will also have transition rules

$\delta(q_1, a, S) = \{(q_1, A)\}$
$\delta(q_1, a, A) = \{(q_1, ABC), (q_1, \lambda)\}$
$\delta(q_1, b, A) = \{(q_1, B)\}$
$\delta(q_1, b, B) = \{(q_1, \lambda)\}$
$\delta(q_1, c, C) = \{(q_1, \lambda)\}$

The sequence of moves made by $M$ in processing $aaabc$ is

$(q_0, aaabc, Z) \vdash (q_1, aaabc, SZ)$
$\vdash (q_1, aabc, AZ)$
$\vdash (q_1, abc, ABCZ)$
$\vdash (q_1, bc, BCZ)$
$\vdash (q_1, c, CZ)$
$\vdash (q_1, \lambda, Z)$
$\vdash (q_f, \lambda, Z)$

This corresponds to the derivation

$S \rightarrow aA$
$S \rightarrow aaABC$
$S \rightarrow aaaBC$
$S \rightarrow aaabc$
$S \rightarrow aaabc$

***Prob.33. Construct an NPDA corresponding to the grammar –***

$$S \to aABB/aAA,\ A \to aBB/a,\ B \to bBB/A$$

***(R.G.P.V., June 2004)***

***Sol.*** The required NPDA $A$ can be defined as follows –

$$A = (\{q\}, \{a, b\}, \{S, A, B, a, b\}, \delta, q, S, \phi)$$

where $\delta$ is defined by the following rules –

$$R_1 - \delta(q, \Lambda, S) = \{(q, aABB), (q, aAA)\}$$
$$R_2 - \delta(q, \Lambda, A) = \{(q, aBB), (q, a)\}$$
$$R_3 - \delta(q, \Lambda, B) = \{(q, bBB), (q, A)\}$$
$$R_4 - \delta(q, a, a) = \{(q, \Lambda)\}$$
$$R_5 - \delta(q, b, b) = \{(q, \Lambda)\}$$

Let us see, the sequence of moves in processing the string *aabaaa*.

$(q, aabaaa, S) \vdash (q, aabaaa, aABB)$ by rule $R_1$
$\vdash (q, abaaa, ABB)$ by rule $R_4$
$\vdash (q, abaaa, aBB)$ by rule $R_2$
$\vdash (q, baaa, BB)$ by rule $R_4$
$\vdash (q, baaa, bBB)$ by rule $R_3$
$\vdash (q, aaa, BBB)$ by rule $R_4$
$\vdash (q, aaa, ABB)$ by rule $R_4$
$\vdash (q, aaa, aBB)$ by rule $R_2$
$\vdash (q, aa, BB)$ by rule $R_4$
$\vdash (q, aa, AB)$ by rule $R_4$
$\vdash (q, aa, aB)$ by rule $R_2$
$\vdash (q, a, B)$ by rule $R_4$
$\vdash (q, a, A)$ by rule $R_3$
$\vdash (q, a, a)$ by rule $R_2$
$\vdash (q, \Lambda, \Lambda)$ by rule $R_4$

Thus,

$$aabaaa \in N(A).$$

**Hence, A is the required NPDA.**

# UNIT 5

## TURING MACHINES – BASICS AND FORMAL DEFINITION, LANGUAGE ACCEPTABILITY BY TM, EXAMPLES OF TM, VARIANTS OF TMs – MULTITAPE TM, NDTM, UNIVERSAL TURING MACHINE, OFF-LINE TMs, EQUIVALENCE OF SINGLE TAPE AND MULTITAPE TMs

***Q.1. Explain turing machine as a mathematical model of computation.***
***(R.G.P.V., June 2003)***

***Or***

***Explain the concept of turing machine model. (R.G.P.V., June 2015)***

***Ans.*** Turing machine (TM) is a simple mathematical model of a general purpose computer. We can say that, the turing machine model is a computing power of the computer, i.e., turing machine is capable of performing any calculation which can be performed by any computing machine.

Turing machine can be thought of as a finite-state automaton connected to a R/W (read/write) head. It has a tape for temporary storage, which is divided into number of cells. Each cell can store only one symbol. The input to and the output from the finite state automaton are effected by the read/write head which can examine one cell at a time. The machine examines the present symbol under the read/write head on the tape, in one move and the present state of an automaton to determine –

(i) A new symbol to be written on the tape in the cell under the read/write head.

(ii) A motion of the read/write head along the tape, which may be either one cell left ($L$) or one cell right ($R$).

(iii) The next state of the automaton.

(iv) Whether to halt or not.

Fig. 5.1 shows an intuitive visualization of a turing machine.

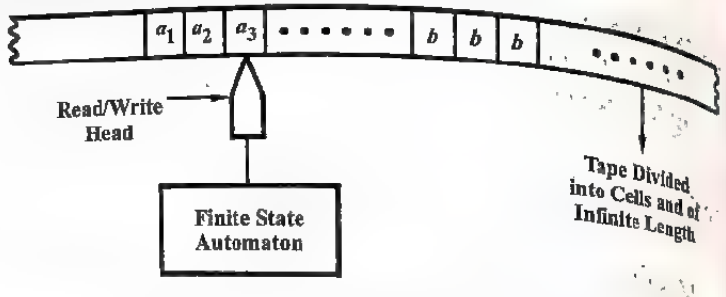



*Fig. 5.1 Turing Machine Model*

***Q.2. Give definition of a turing machine.***

***Ans.*** More clearly, a turing machine $M$ is defined as –

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F),$$

where

$Q$ is the set of internal states

$\Sigma$ is the set of input alphabet

$\Gamma$ is a finite set of symbol known as ***tape alphabet***

$\delta$ is the transition function

$B \in \Gamma$ is a special symbol called ***blank***

$q_0 \in Q$ is the initial state

$F \subseteq Q$ is the set of final states.

In the definition of a turing machine, we assume that $\Sigma \subseteq \Gamma - \{B\}$, i.e. the input alphabet is a subset of the tape alphabet, not including the blank. Blanks are ruled out as input.

The transition function $\delta$ is defined as follows –

$$\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R\}.$$

In general, $\delta$ is a partial function on $Q \times \Gamma$. Its interpretation gives the principle by which a turing machine operate. The arguments of $\delta$ are the current state of the control unit and the current tape symbol being read. The output is a new state of the control unit, a new tape symbol, which replaces the previous one, and a move symbol, $L$ or $R$. The move symbol indicates whether the read-write head moves left or right one cell after the new symbol has been written on the tape.

***Q.3. Give similarities and differences of turing machines and finite state automata.*** ***(R.G.P.V., Dec. 2008)***

***Or***

***Differentiate the purpose of the study of turing machine with finite automata/pushdown automata.*** ***(R.G.P.V., Dec. 2013)***



***Ans.*** **Differences** – (i) A finite state automaton is described by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, while a turing machine has a 7-tuple, viz. $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$.

(ii) A finite automaton has only read head, while a turing machine has R/W head.

**Similarities** – (i) The turing machine can be thought of as a finite state automaton connected to a R/W head.

(ii) Both have a finite control, an input tape that is divided into cells, and a tape head that scans one cell of the tape at a time.

***Q.4. Explain TM and its parameter.*** ***(R.G.P.V., June 2011)***

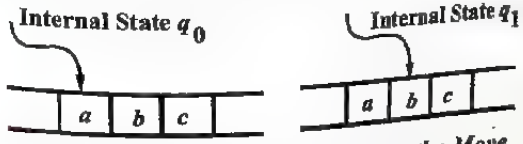
***Ans.*** Refer to Q.1 and Q.2.

***Q.5. Turing machine models the computing capacity of a general purpose computer. Justify with complete example.*** ***(R.G.P.V., June 2010)***

***Ans.*** Fig. 5.2 shows the situation before and after the move caused by the transition

$$\delta(q_0, a) = (q_1, b, R).$$

We can consider a turing machine as a rather simple computer. It has a processing unit, which has a finite memory, and in its tape, it has a secondary storage of unlimited capacity. The instructions that such a computer (i.e., turing machine) can carry out are very limited, it can sense a symbol on its tape and use the result to take decision what to do next. The actions a turing machine can perform are to rewrite the current symbol, to change the state of the control, and to move the read-write head. This small instruction set may look inadequate for doing complicated things, but this is not so. Turing machines are much powerful in principle. The transition function $\delta$ define how this computer acts, and we often call it the "program" of the machine.




***(a) Before the Move*** ***(b) After the Move***

*Fig. 5.2*

As always, the automaton starts in the given initial state having some information on the tape. It then goes through a sequence of steps controlled by the transition function ($\delta$). During this process, the contents of any cell on the tape may be examined and changed as many times as it need. Eventually, the whole process may terminate, which we achieve in a turing machine by putting it into a ***halt state***. A Turing machine is said to be in halt state, whenever

it reaches a configuration for which $\delta$ is not defined, this is possible because $\delta$ is a partial function. In fact, we will assume that no transitions are defined for any final state, so the turing machine will halt whenever it comes in final state.

Let us take an example –

Consider the turing machine defined by –

$$Q = \{q_0, q_1\}$$
$$\Sigma = \{a,b\}$$
$$\Gamma = \{a, b, B\}$$
$$F = \{q_1\}$$

and

$$\delta(q_0, a) = (q_0, b, R)$$
$$\delta(q_0, b) = (q_0, b, R)$$
$$\delta(q_0, B) = (q_1, B, L)$$

If this turing machine is started in state $q_0$ with the input symbol $a$ under the read-write head, the applicable transition rule is $\delta(q_0, a) = (q_0, b, R)$. So, the read-write head will replace the $a$ with $b$, then move right on the tape. The machine will still remain in state $q_0$. Any subsequent $a$ will also be replaced with $b$, but $b$'s will not be modified. When the machine finds the first blank, it will move left one cell, then half in final state $q_1$. Fig. 5.3 shows a sequence of moves and several stages of the process for a simple initial configuration.

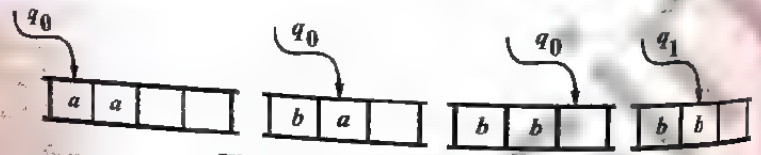



*Fig. 5.3 A Sequence of Moves*

***Q.6. Explain representation of turing machines with the help of examples.***

***Ans.*** We can describe a turing machine using – (i) Instantaneous description using move-relations, (ii) Transition table, and (iii) Transition diagram.

***(i) Representation by Instantaneous Description*** – 'Snapshots' of a turing machine in action can be used to describe a turing machine. These give "instantaneous description" of a turing machine. An ID of a Turing machine is defined in terms of the entire input string and the current state.

An ID of a turing machine $M$ is a string $\alpha\beta\gamma$, where $\beta$ is the present state of $M$, the entire input string is split as $\alpha\gamma$, the first symbol of $\gamma$ is the current symbol, say $a$, under R/W head and $\gamma$ has all the subsequent symbols of the input string, and the string $\alpha$ is the substring of the input string formed by all the symbols to the left of $a$.



A snapshot of turing machine is shown in fig. 5.4 and the corresponding ID is given in fig. 5.5.

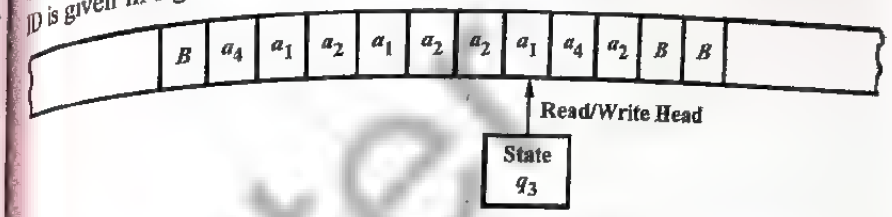



*Fig. 5.4 Snapshot of Turing Machine*

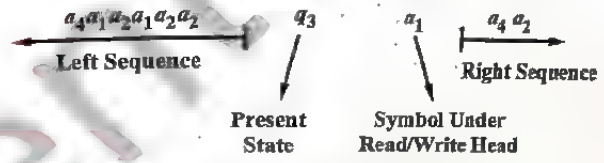



*Fig. 5.5 Representation of ID*

**Moves in a TM** – $\delta(q, x)$ induces a change in ID of the turing machine. This change in ID is called a move.

For example – Suppose $\delta(q, x_i) = (p, y, L)$. The input string to be processed is $x_1 x_2 \ldots x_n$, and the present symbol under R/W head is $x_i$. So the ID before processing $x_i$ is

$$x_1 x_2 \ldots x_{i-1}\, q x_i \ldots x_n$$

After processing of $x_i$, the resulting ID is

$$x_1 x_2 \ldots x_{i-2}\, p x_{i-1}\, y x_{i+1} \ldots x_n$$

This change in ID is represented by

$$x_1 x_2 \ldots x_{i-1}\, q x_i \ldots x_n \vdash x_1 x_2 \ldots x_{i-2}\, p x_{i-1}\, y x_{i+1} \ldots x_n$$

If $\delta(q, x_i) = (p, y, R)$, then the change of ID is represented by

$$x_1 x_2 \ldots x_{i-1} q x_i \ldots x_n \vdash x_1 x_2 \ldots x_{i-1}\, y p x_{i+1} \ldots x_n.$$

We can denote an ID by $I_j$ for some $j$. $I_j \vdash I_k$ defines a relation among IDs. So the symbol $\vdash^*$ denotes the reflexive-transitive closure of the relation $\vdash$. If $I_1 \vdash^* I_n$, then we can split this as $I_1 \vdash I_2 \vdash \ldots \vdash I_n$ for some IDs, $I_2, \ldots, I_{n-1}$.

It is noted that the description of moves, by IDs is very much useful to represent the processing of input strings.

***(ii) Representation by Transition Table*** – The definition of $\delta$ can be given in the form of a table called the transition table. If $\delta(q, a) = (\gamma, \alpha, \beta)$ we write $\alpha\beta\gamma$ under $a$-column and $q$-row. So if we get $\alpha\beta\gamma$ in the table, it

means that $\alpha$ is written in the current cell, $\beta$ gives movement of the head (L or R) and $\gamma$ denotes the new state into which the turing machine enters.

For example, consider a turing machine with five states $q_1, \ldots, q_5$, where $q_1$ is the initial state and $q_5$ is the final state. The tape symbols are 0, 1, and B. The transition table given in table 5.1 describes $\delta$.

**Table 5.1 Transition Table of a Turing Machine**

| *Present State* | *Tape Symbol* | | |
|---|---|---|---|
| | ***B*** | ***0*** | ***1*** |
| $\rightarrow q_1$ | $1Lq_2$ | $0Rq_1$ | |
| $q_2$ | $BRq_3$ | $0Lq_2$ | $1Lq_2$ |
| $q_3$ | | $BRq_4$ | $BRq_5$ |
| $q_4$ | $0Rq_5$ | $0Rq_4$ | $1Rq_4$ |
| $(q_5)$ | $0Lq_2$ | | |

***(iii) Representation by Transition Diagram*** – Turing machine can be represented by transition system. The states are represented by vertices. Directed edges are used to represent transition of states. The labels are triples of the form $(\alpha, \beta, \gamma)$, where $\alpha, \beta \in \Gamma$ and $\gamma \in \{L, R\}$. When there is a directed edge from $q_i$ to $q_j$ with label $(\alpha, \beta, \gamma)$, it means that

$$\delta(q_i, \alpha) = (q_j, \beta, \gamma)$$

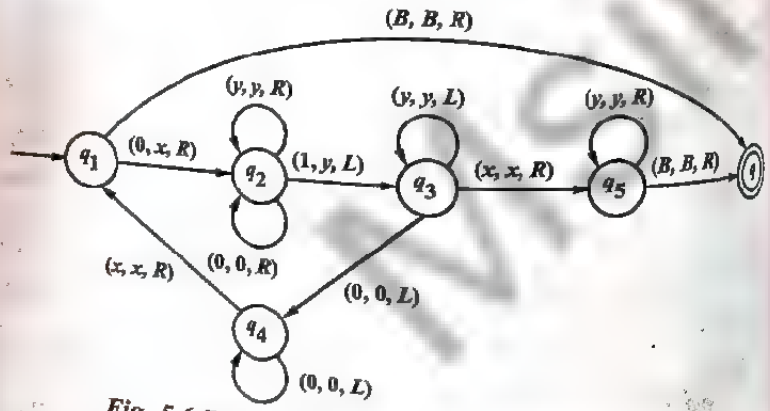



***Fig. 5.6 Transition Diagram of a Turing Machine***

During the processing of an input string, suppose the turing machine enters $q_i$ and R/W head scans the (present) symbol $\alpha$. As a result, the symbol $\beta$ is written in the cell under R/W head. The R/W head moves to the left or the right, depending on $\gamma$, and the new state is $q_j$.



Every edge in the transition system can be represented by a 5-tuple $(q_i, \alpha, \beta, \gamma, q_j)$. So each turing machine can be described by the sequence of 5-tuples representing all the directed edges. The initial state is indicated by $\rightarrow$ and any final state is marked with circle.

For example, a turing machine represented by the transition system is given in fig. 5.6.

***Q.7. Explain ID of a turing machine.*** ***(R.G.P.V., June 2016)***

***Ans.*** Refer to Q.6.

***Q.8. Give the basic guidelines for designing a turing machine. Also give an example.***

***Or***

***Write brief note on design of TM.*** ***(R.G.P.V., Dec. 2009)***

***Ans.*** The basic guidelines for designing a turing machine are as follows –

(i) The fundamental objective in scanning a symbol by R/W head is to 'know' what to do in the future. The machine must remember the past symbols scanned. The turing machine can remember this by going to the next unique state.

(ii) The number of states must be minimized. This can be achieved by changing the states only when there is a change in the written symbol or when there is a change in the movement of R/W head. We explain the design by a simple example.

We design a turing machine to recognize all strings consisting of even number of 1's.

The construction is made by defining moves in the following manner –

(i) $q_1$ is the initial state. $M$ enters state $q_2$ on reading 1 and writes $B$.

(ii) If $M$ is in state $q_2$ and reads 1, it enters $q_1$ and writes $B$.

(iii) $q_1$ is the only accepting state.

So $M$ accepts a string if it exhausts all input symbols and finally in state $q_1$. Symbolically,

$$M = (\{q_1, q_2\}, \{1\}, \{1, B\}, \delta, q_1, B, \{q_1\})$$

where $\delta$ is defined by the table 5.2.

**Table 5.2 Transition Table**

| *Present State* | *Input 1* |
|---|---|
| $\rightarrow (q_1)$ | $Bq_2 R$ |
| $q_2$ | $Bq_1 R$ |

Let us obtain the sequence of moves for string 11. Thus, $q_1 11 \vdash Bq_2 1$ $\vdash BBq_1$. As $q_1$ is the final state, 11 is accepted. $q_1 111 \vdash Bq_2 11 \vdash BBq_2 1$ $\vdash BBBq_2$. $M$ halts in state $q_2$ and $q_2$ is not an accepting state. Thus, 111 is not accepted by $M$.

***Q.9. What is turing-computable function ? Define recursive function.***
***(R.G.P.V., Dec. 2014)***

***Ans.*** A function $f$ with domain $D$ is said to be turing-computable or just computable if there exists some turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ such that

$$q_0 w \overset{*}{\underset{M}{\vdash}} q_f f(w)$$

for all $w \in D$,

The class of recursive functions include the class of primitive recursive functions.

Let $g(a_1, a_2, \ldots\ldots, a_n)$ be a total function over $N$. $g$ is regular function if there exists some natural number $b_0$ such that $g(a_1, a_2, \ldots\ldots, a_n, b_0) = 0$ for all values $a_1, a_2, \ldots\ldots, a_n$ in $N$.

For instance, $g(a, b) = \min(a, b)$ is a regular function since $g(a, 0) = 0$ for all $a$ in $N$. But $f(a, b) = |a - b|$ is not regular since $f(a, b) = 0$ only when $a = b$, and so we cannot find a fixed b such that $f(a, b) = 0$ for all $a$ in $N$.

A function $f(a_1, a_2, \ldots\ldots, a_n)$ over $N$ is defined from a total function $g(a_1, a_2, \ldots\ldots, a_n, b)$ by minimization if –

(i) $f(a_1, a_2, \ldots\ldots, a_n)$ is the least value of all $b$'s such that $g(a_1, a_2, \ldots\ldots, a_n, b) = 0$ if it exists. The least value is denoted by $\mu b[g(a_1, a_2, \ldots\ldots, a_n, b) = 0]$,

(ii) $f(a_1, a_2, \ldots\ldots, a_n)$ is undefined if there is no $b$ such that $g(a_1, a_2, \ldots\ldots, a_n, b) = 0$.

It is noted that, $f$ is partial. But, if $g$ is regular then $f$ is total.

A function is recursive if it can be obtained from the initial functions by a finite number of applications of composition, recursion and minimization over regular functions.

A function is partial recursive if it can be obtained from the initial functions by a finite number of applications of composition, recursion and minimization.

***Q.10. Give definition of a turing machine. What type of language can it accept ? Types of turing machine.*** ***(R.G.P.V., June 2016)***

***Ans.*** **Definition** – Refer to Q.2.

**Type of Language that a Turing Machine can Accept** – Turing machines can be viewed as accepters in the following sense –



A string $w$ is written on the tape, with blanks filling out the unused portion. The machine is started in the initial state $q_0$ with the read-write head positioned on the leftmost symbol of $w$. If, after a sequence of moves, the turing machine enters a final state and halts, then $w$ is considered to be accepted.

Formally, we can say that –

If $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ is a turing machine. Then the language accepted by $M$ is

$$L(M) = \{w \in \Sigma^+ : q_0 w \overset{*}{\vdash} x_1 q_f x_2 \text{ for some } q_f \in F, x_1, x_2 \in \Gamma^*\}.$$

The above said definition indicates that the input $w$ is written on the tape with blanks on either side. The reason for excluding blanks from the input now becomes clear, i.e., it assures us that all the input restricted to a well-defined region of the tape, bracketed by blanks on the right and left side. Without this convention, the machine could not limit the region in which it must look for the input, no matter how many blanks it saw, it could never be sure that there was not some non-blank input somewhere else on the tape.

The definition tells us what must happen when $w \in L(M)$. It says nothing about the outcome for any other input. When $w$ is not in $L(M)$, one of two things can happen, i.e., the machine can halt in a non-final state or it can enter an infinite loop and never halt. Any string for which $M$ does not halt is by definition not in $L(M)$.

**Types of Turing Machine** – Refer to Q.23.

***Q.11. Explain how turing machine can be used as generating device.***
***(R.G.P.V., June 2009)***

***Ans.*** Turing machine can be used as a generating device, generating strings over some alphabet, in addition to using it as a recognizer of language and computer of integer functions. A turing machine can be used as generator of strings as follows –

Consider a turing machine with multiple tapes. One of these tapes is used as output tape, which has the special property, that once a symbol is written on it, it cannot be changed, and tape head for this tape never moves to left. If the moves of this turing machine are designed in such a manner that it writes some string over some alphabet $\Sigma$ on its output tape, separated by marker symbol #. Then set of all those strings that it can eventually write on its output tape between pair of #'s constitutes the language generated by it. Therefore, we formally define the language generated by the turing machine $M$ as –

$G(M) = \{w | w$ is in $\Sigma^*$, and $w$ is eventually written on the output tape between pair of #'s$\}$. For example consider a turing machine with following moves –

(i) It starts in the initial with blank tape, and replaces the blank symbol scanned by tape head with #, and enter into a state $q_1$, and moves tape head right.

(ii) In $q_1$ it replaces the next blank by 0, enters into state $q_2$, and moves the tape head right.

(iii) In state $q_2$, it either replaces the next blank by 1 or 0, and remains in state $q_2$, or replaces next blank by 1, and enters into state $q_3$, in both cases the tape head is moved right.

(iv) In state $q_3$, it replaces the next blank by 1, and enters into state $q_4$, moves tape head right.

(v) In state $q_4$, it replaces the next blank by #, and enters into state $q_0$.

The above turing machine, therefore, prints those strings that starts with 0, and ends will 11. Hence, the turing machine generates the language given by the regular expression 0(0|1)*11.

The moves of the above turing machine can be represented by the following transition diagram shown in fig. 5.7.

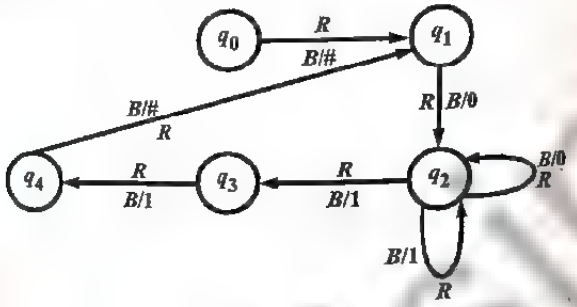



*Fig. 5.7*

***Q.12. What do you know about multitape turing machine ? Compare it with the standard turing machine.***

***Or***

***Explain the multitape turing machine.*** ***(R.G.P.V., Dec. 2007)***

***Ans.*** A multitape turing machine consists of a finite control with several tape with its own controlled R/W heads as shown in fig. 5.8.

Each tape is infinite in both direction. On a single move depending upon the state of the finite control and the sym- bol scanned by each of the tape heads, the machine can –

(i) Change state

(ii) Print a new symbol on each of the cells scanned by its tape heads

(iii) Move each of its tape heads, independently, one cell to the left or right, or keep it stationary.



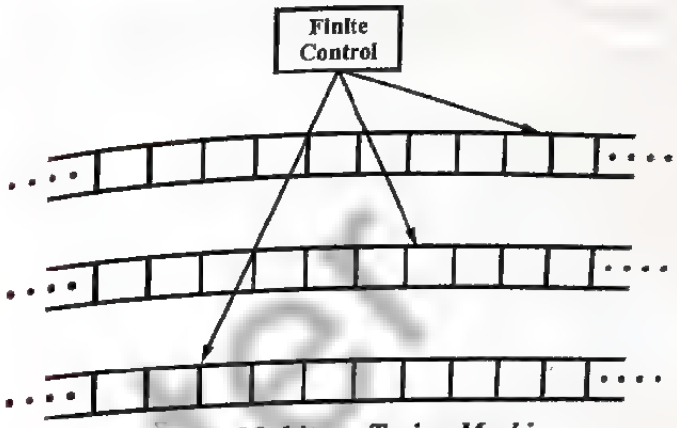



*Fig. 5.8 Multitape Turing Machine*

Initially, the input appears on the first tape, and the other tapes are blank.

The formal definition of a multitape TM goes beyond the definition of a standard TM, since it requires a modified transition function. Typically, we define an $n$-tape machine by $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, where $Q, \Sigma, \Gamma, q_0, B$ and $F$ are as in the definition of a turing machine and the transition function $\delta$ is defined as –

$$\delta : Q \times \Gamma^n \to Q \times \Gamma^n \times \{L, R\}^n$$

$\delta$ specifies what happens on all the tapes. For example, if $n = 2$, with a current configuration shown in fig. 5.9 (a), then

$$\delta(q_0, a, e) = (q_1, x, y, L, R)$$

is interpreted as follows –

The transition rule can be applied only if the machine is in state $q_0$ and the first R/W head encounters an $a$ and the second an $e$. The symbol on the first tape will then be replaced with an $x$ and its R/W head will move towards left. At the same time, the symbol on the second tape is rewritten as $y$ and the R/W head moves right. The control unit then changes its state to $q_1$ and the machine goes into the new configuration as shown in fig. 5.9(b).

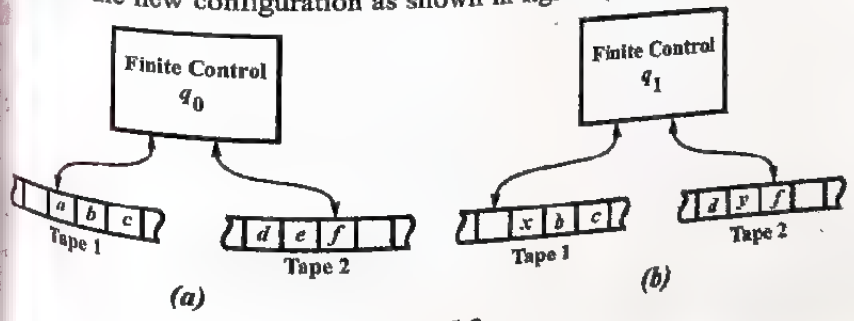



*Fig. 5.9*

***Q.13. What do you mean by turing machine ? Explain multiple tapes turing machine.*** ***(R.G.P.V., Dec. 2013, Nov. 2018)***

***Ans.*** **Turing Machine** – Refer to Q.2.

**Multiple Tapes Turing Machine** – Refer to Q.12.

***Q.14. Explain non-deterministic turing machine.*** ***(R.G.P.V., Dec. 2007)***

***Ans.*** A non-deterministic turing machine is an automaton as given by the definition of a standard turing machine

$$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$$

As always when non-determinism is involved, the range of $\delta$ is a set of possible transitions, any of which can be chosen by the machine.
For example –

If a turing machine has transitions specified by

$$\delta(q_0, a) = \{(q_1, b, R), (q_2, c, L)\}$$

it is non-deterministic. The moves

$$q_0aaa \vdash bq_1aa \text{ and } q_0aaa \vdash q_2Bcaa$$

are both possible.

As it is not clear, what role, non-determinism plays in computing functions, non-deterministic automata are usually viewed as acceptors. But a non-deterministic turing machine is said to accept $u$ if there is any possible sequence of moves such that

$$q_0u \vdash^* x_1q_fx_2,$$

with $q_f \in F$. A non-deterministic machine may have moves available which lead to a non-final state or to an infinite loop. But, as always with non-determinism, these alternatives are irrelevant. All we are interested in is the existence of some sequence of moves leading to acceptance.

***Q.15. Show that if L is accepted by a non-deterministic turing machine $M_1$, then L is accepted by some deterministic turing machine $M_2$.***

***Ans.*** For any state and tape symbol of $M_1$, there is a finite number of choices for the next move. These can be numbered 1, 2, 3,..... Let $n$ be the maximum number of choices for any state-tape symbol pair. Then any finite sequence of choices can be represented by a sequence of the digits 1 through $n$. Not all such sequences may represent choices of moves, since there may be fewer than $n$ choices in some situations.

$M_2$ will have three tapes. The first hold the input. $M_2$ will generate sequences of the digits 1 through $n$ in a systematic manner. Specifically, the sequences will be generated with the shortest appearing first. Sequences of equal length are generated in numerical order.



For each sequence generated on tape 2, $M_2$ copies the input onto tape 3 and then simulate $M_1$ on tape 3, using the sequence on tape 2 to dictate the moves of $M_1$. If $M_1$ enters an accepting state, $M_2$ also accepts. If there is a sequence of choices leading to acceptance, it will eventually be generated on tape 2. When simulated, $M_2$ will accept. But if no sequence of choices of moves of $M_1$ leads to acceptance, $M_2$ will not accept.

***Q.16. What are the features of universal turing machine ?*** ***(R.G.P.V., Dec. 2014)***

***Ans.*** The features of universal turing machine are as follows –

(i) Universal turing machine can simulate any other turing machine.

(ii) Universal turing machine has an ability to manipulate an unbounded amount of data in finite amount of time.

***Q.17. What do you mean by universal turing machine ?***

***Or***

***Write a note on universal turing machine.*** ***(R.G.P.V., Dec. 2009)***

***Or***

***How UTM overcomes the limitation of Turing machine ? Also define UTM.*** ***(R.G.P.V., Dec. 2015)***

***Ans.*** Consider the following argument against Turing thesis – A turing machine is a special purpose computer. Once $\delta$ is defined the machine is restricted to carrying out one particular type of computation. Digital computers, on the other hand, are general purpose machines that can be programmed to do different jobs at different times. Consequently, turing machines cannot be considered equivalent to general purpose digital computers. This objection can be overcome by designing a reprogrammable turing machine called a universal turing machine.

A ***universal turing machine*** $M_u$ is an automaton that, given as input the description of any turing machine $M$ and a string $u$, can simulate the computation of $M$ on $u$. To construct such an $M_u$ we first select a standard way of describing turing machines. We may, without loss of generality, assume that,

$$Q = \{q_1, q_2, \ldots\ldots, q_n\}$$

with $q_1$ the initial state, $q_2$ the only final state, and

$$\Gamma = \{a_1, a_2, \ldots\ldots, a_m\}$$

where $a_1$ represents the blank. Then we pick an encoding in which $q_i$ is represented by 1, $q_2$ is represented by 11, and so on. Similarly, $a_i$ is encoded as 1, $a_2$ as 11, and so on. The symbol 0 will be used as a separator between the 1's. With the initial and final state and the blank defined by this convention,

any turing machine can be explained completely with $\delta$ only. The transition function is encoded according to this scheme, with the arguments and result in some prescribed sequence. For example, $\delta(q_1, a_2) = (q_2, a_3, L)$ might appear as

... 10110110111010...

It means that any turing machine has a finite encoding as a string on $\{0,1\}^+$, and that, given any encoding of $M$, we can decode it uniquely. Some strings will not represent any turing machine (for example, the string 00011), but we can easily spot these, so they are of no concern.

A universal turing machine $M_u$ then has an input alphabet that includes $\{0,1\}$ and the structure of a multitape machine, as shown in fig. 5.10.

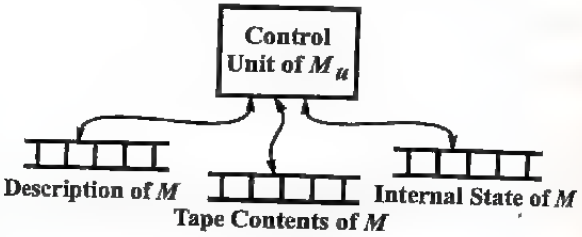



*Fig. 5.10*

For any input $M$ and $u$, tape 1 will have an encoded definition of $M$, and tape 2 will contain the tape contents of $M$, and tape 3 the internal state of $M$. $M_u$ sees first at the contents of tapes 2 and 3 to find the configuration of $M$. Then it consults tape 1 to see what $M$ would do in this configuration of $M$. Finally, the tapes 2 and 3 will be modified to reflect the result of the move.

***Q.18. Explain multitape and universal turing machine.***

***(R.G.P.V., June 2016)***

***Ans.*** Refer to Q.12 and Q.17.

***Q.19. Write short notes –***

***(i) Turing machine***

***(ii) Universal turing machine***

***(R.G.P.V., Dec. 2017)***

***Ans. (i) Turing Machine*** – Refer to Q.1.

***(ii) Universal Turing Machine*** – Refer to Q.17.

***Q.20. Formally define an off-line turing machine. Also give the arguments that any language accepted by an off-line turing machine is also accepted by some standard machine.***



***Or***

***Write a short note on off-line turing machine.***

***(R.G.P.V., June 2003, 2004)***

***Ans.*** An off-line turing machine is a machine in which each move is governed by the internal state, what is currently read from the input file and what is seen by the R/W head. A schematic representation of an off-line turing machine is shown in fig. 5.11.

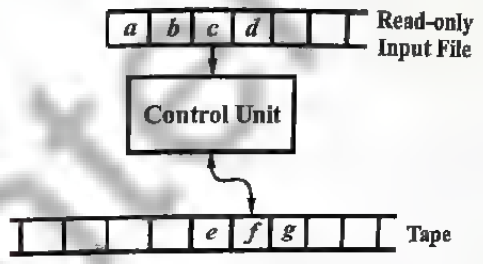



*Fig. 5.11*

The behaviour of any standard turing machine can be simulated by some off-line model. All that needs to be done by the simulating machine is to copy the input from the input file to the tape. Then it can proceed in the same way as the standard machine.

The simulation of an off-line machine $M$ by a standard machine $M'$ requires a lengthy description. A standard machine can simulate the computations of an off-line machine by using the four-track arrangement shown in fig. 5.12. In which, the tape contents represent the specific configuration of fig. 5.11. Each of the four tracks of $M'$ plays a specific role in the simulation. The first track has the input, the second marks the position where the input is read, the third represents the tape of $M$, and the fourth shows the position of $M$'s R/W head.

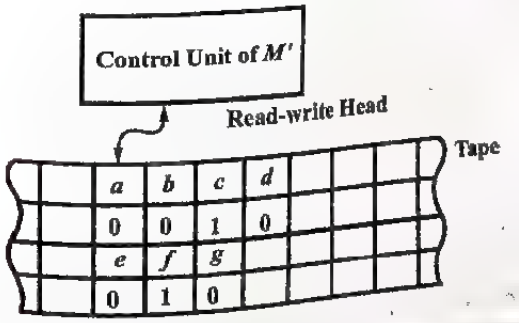



*Fig. 5.12*

The simulation of each move of $M$ requires a number of moves of $M'$. Beginning from some standard position, say the left end, and with the relevant information marked by special end markers, $M'$ seeks track 2 to locate the position where the input file of $M$ is read. The symbol found in the corresponding cell on track 1 is remembered by putting the control unit of $M'$ into a state chosen for this purpose. Next, track 4 is searched for the position of the R/W head of $M$. With the remembered input and the symbol on track 3, we now know what $M$ is to do. This information is again remembered by $M'$ with an appropriate internal state. Next, all four tracks of simulating machine's tape are modified to reflect the move of $M$,. Finally, the R/W head of $M'$ returns to the standard position for the simulation of the next move.

***Q.21. Write a brief note on multidimensional turing machine.***

***(R.G.P.V., Dec. 2002)***

***Ans.*** Multidimensional turing machine is one in which the tape can be viewed as extending infinitely in more than one dimension. A diagram of a two-dimensional turing machine is shown in fig. 5.13 (a).

The formal definition of a two-dimensional turing machine involves a transition function $\delta$ of the following form –

$$\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R, U, D\},$$

where $U$ and $D$ specify movement of the R/W head up and down, respectively.

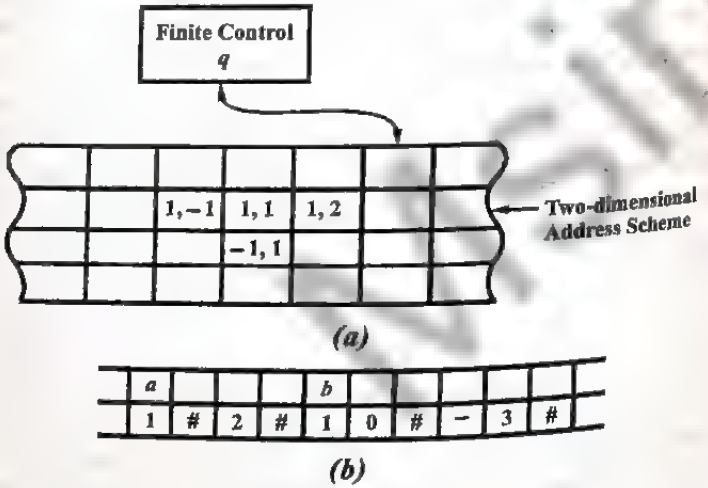



***Fig. 5.13***

To simulate this machine on a standard turing machine, we can use the two-track model depicted in fig. 5.13 (b). First, we associate an ordering of address with the cells of the two-dimensional tape. This can be done in a number of ways, for example, in two-dimensional fashion indicated in fig.



5.13 (a). The two-track tape of the simulating machine will use one track to store cell contents and the other one to keep the associated address. In the scheme of fig. 5.13 (a), the configuration in which cell (1, 2) contains $a$ and cell (10, – 3) contains $b$ is shown in fig. 5.13 (b). Note one complication – the cell address can involve arbitrarily large integers, so the address track cannot use a fixed-size field to store address. Instead, we must use a variable field-size arrangement, using some special symbols to delimit the fields, as shown in fig. 5.13.

Let us suppose that, at the start of the simulation of each move, the R/W head of the two-dimensional machine $M$ and the R/W head of the simulating machine $M'$ are always on the corresponding cells. To simulate a move, the simulating machine, i.e., $M'$, first compute the address of the cell to which $M$ is to move. Using the two-dimensional address scheme, it is a simple computation. Once the address is computed, simulating machine $M'$ finds the cell with this address on track 2 and then change the cell contents to account for the move of $M$. Again, given $M$, there is a straight forward construction for $M'$.

***Q.22. Write brief note on the two-way infinite tape TM.***

***(R.G.P.V., Dec. 2009)***

***Ans.*** A turing machine with a two-way infinite tape is denoted by $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, as in the original model. As its name implies, the tape is infinite to the left as well as to the right. We denote an ID of such a device as for the one-way infinite TM. We imagine, however, that there is an infinity of blank cells both to the left and right of the current non-blank portion of the tape,

The relation $\vdash_M$, which relates two ID's if the ID on the right is obtained from the one on the left by a single move, is defined as for the original model with the exception that if $\delta(q, X) = (p, Y, L)$, then $qX\alpha \vdash_M pBY\alpha$ (in the original model, no move could be made), and if $\delta(q, X) = (p, B, R)$, then $qX\alpha \vdash p\alpha$ (in the original, the $B$ would appear to the left of $p$).

The initial ID is $q_0w$. While there was a left end to the tape in the original model, there is no left end of the tape for the Turing machine to "fall off", so it can proceed left as far as it wishes. The relation $\vdash_M^*$, as usual, relates two ID's if the one on the right can be obtained from the one on the left by some number of moves.

***Q.23. Write short note on various types of turing machine.***

***(R.G.P.V., Dec. 2004, June 2006)***

***Or***

***Explain multitape and multi-head turing machine.***

***(R.G.P.V., Dec. 2011)***

**Or**

***Write short note on models of turing machine.*** **(R.G.P.V., Dec. 2006)**

**Or**

***Explain the types of turing machine.*** **(R.G.P.V., Dec. 2016)**

***Ans.*** The various types of turing machines are as follows –

(i) Multitape turing machine
(ii) Multidimensional turing machine
(iii) Off-line turing machine
(iv) Multiple tracks turing machine
(v) Multiple heads turing machine
(vi) Universal turing machine.

***(i) Multitape Turing Machine*** – Refer to Q.12.

***(ii) Multidimensional Turing Machine*** – Refer to Q.21.

***(iii) Off-line Turing Machine*** – Refer to Q.20.

***(iv) Multiple Tracks Turing Machine*** – We know that, a turing machine $M$ is defined by

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F),$$

where, $Q$ is the set of internal states
$\Sigma$ is the set of input alphabet
$\Gamma$ is a finite set of symbols called the tape alphabet
$\delta$ is the transition function
$B \in \Gamma$ is a special symbol called the blank
$q_0 \in Q$ is the initial state
$F \subseteq Q$ is the set of final states.

By the above definition, it is implicit that each tape symbol can be a composite of characters rather than just a single one. This can be made more explicit by fig. 5.14, in which the tape symbols are triplets from some simpler alphabet.

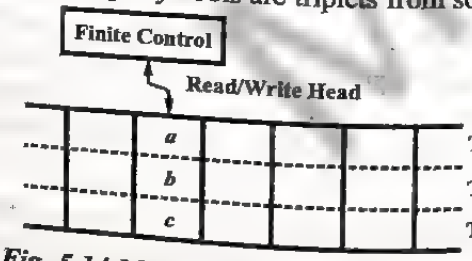



***Fig. 5.14 Multiple Tracks in Turing Machine***

In fig. 5.14, we have divided each cell of the tape into three parts, which are known as tracks, and each containing one member of the triplet



based on this visualization of fig. 5.14, an automaton is sometimes called a turing machine with multiple tracks, but such a view is not required for being modification in the definition of the turing machine, since all we need to do is make $\Gamma$ an alphabet in which each symbol is composed of several parts.

***(v) Multiple Heads Turing Machine*** – If we allow a turing machine to have one tape, but several heads on it, then we say in one step, the all heads sense the scanned symbols and move or write independently. What happen when two heads, that happen to be scanning the same tape square, attempt to write different symbols, for this same convention must be adopted perhaps the head with the lower number wins out. Also, let us assume that the head cannot sense each other's presence in the same tape square, except perhaps indirectly, through unsuccessful writes.

It is not hard to see that a simulation like the one we used for $n$-tape machines can be carried out for turing machines with several heads on a tape. The basic idea is to divide the tape into tracks, all but one of which are used solely to record the head positions. To simulate one computational step by the multiple-head machine, the tape must be scanned twice, once to find the symbols at the head position and next to change those symbols or move the head as appropriate. The number of steps needed is quadratic.

The use of multiple heads, like multiple tapes, can sometimes drastically simplify the construction of a turing machine. A two-head version of the copying machine could function in much more natural than the one-head version or even the two-tape version machine.

***(vi) Universal Turing Machine*** – Refer to Q.17.

***Q.24. What do you understand by composite and iterated turing machines ?***

**Or**

***Differentiate between composite and iterated TM.*** **(R.G.P.V., June 2011)**

***Ans.*** Sometimes, to obtain the solution we may combine two or more turing machines. T.M. will be combined in such a way that output of the first turing machine is input to the second, the output of the second T.M. is input to the third and so on. Also each of the turing machine is solving a smaller problem. This is known as composite turing machine.

For example, we can combine two turing machines to convert into the decimal form of the GCD of two unary numbers. For realizing a composite turing machine (CTM), the FM's of the component T.M's are appropriate, state rather the halt state at the completion of the performance of each component T.M.

Yet another way of having a combination of turing machine is by applying its own output as the next inputs.

The basic idea behind the composite and iterated turing machine is that we can design a more powerful turing machine which can solve a more complicated problem. The concept of splitting the problem into numbers of smaller problems and designing the turing machine has influenced the current computing.

***Q.25. Explain Church's hypothesis.***

***(R.G.P.V., June 2011, Dec. 2012, June 2016)***

***Or***

***Write a brief note on Church's hypothesis.*** ***(R.G.P.V., Dec. 2002)***

***Ans.*** The assumption that the intuitive notion of "computable function" can be identified with the class of partial recursive functions is known as Church's hypothesis or the Church-Turing thesis. While we cannot hope to "prove" Church's hypothesis as long as the informal notion of "computable" remains an informal notion, we can give evidence for its reasonableness. As long as our intuitive notion of "computable" places no bound on the number of steps or the amount of storage, it would seem that the partial recursive functions are intuitively computable, although some would argue that a function is not "computable" unless we can bound the computation in advance or at least establish whether or not the computation eventually terminates.

What is less clear is whether the class of partial recursive functions includes all "computable" functions. Logicians have presented many other formalisms such as the $\lambda$-calculus, post systems, and general recursive functions. All have been shown to define the same class of functions, i.e., the partial recursive functions. In addition, abstract computer models, such as the random access machine (RAM), also give rise to the partial recursive functions.

Note that a turing machine can simulate a RAM, provided that the elementary RAM instructions can themselves be simulated by a TM.

***Q.26. Show that if a language L is accepted by a multitape turing machine, then it is also accepted by a single-tape turing machine.***

***Ans.*** Let $L$ be accepted by $M_1$, a turing machine with $k$-tapes. We can construct $M_2$, a one-tape turing machine with $2k$ tracks, two tracks for each $M_1$'s tape. One track records the contents of the corresponding tape of $M_1$ and other is blank, except for a marker in the cell that holds the symbol scanned



by the corresponding head of $M_1$, as shown in fig. 5.15. The finite control of $M_2$ stores the state of $M_1$, along with a count of the number of head markers to the right of $M_2$'s tape head.

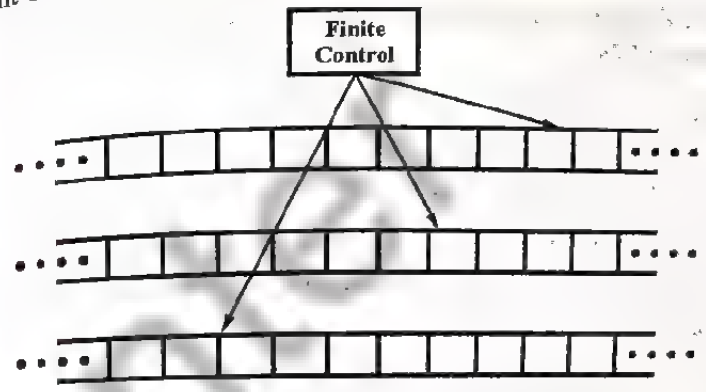



***Fig. 5.15 Multitape Turing Machine***

Each move of $M_1$ is simulated by a sweep from left to right and then from right to left by the tape head of $M_2$. Initially, $M_2$'s head is at the leftmost cell containing a head marker. To simulate a move of $M_1$, $M_2$ sweeps right, visiting each of the cells with head marker and recording the symbol scanned by each head of $M_1$.

| | | | | | | |
|---|---|---|---|---|---|---|
| **Head 1** | | $X$ | | | | |
| **Tape 1** | $A_1$ | $A_2$ | ... | | ... | $A_m$ |
| **Head 2** | | | | $X$ | | |
| **Tape 2** | $B_1$ | $B_2$ | .... | | ... | $B_m$ |
| **Head 3** | $X$ | | | | | |
| **Tape 3** | $C_1$ | $C_2$ | ... | | ... | $C_m$ |

***Fig. 5.16 Simulation of Three Tapes by One***

When $M_2$ crosses a head marker, it must update the count of head marker to its right. When no more head marker are to the right, $M_2$ has seen the symbols scanned by each of $M_1$'s heads, so $M_2$ has enough information to determine the move of $M_1$. Now, $M_2$ makes a pass left, until it reaches the leftmost head marker. The count of markers to the right enables $M_2$ to tell when it has gone far enough. As $M_2$ passes each head marker on the leftward pass, it updates the tape symbol of $M_1$ "scanned" by the head marker, and it moves the head marker one symbol left or right to simulate the move of $M_1$. Finally, $M_2$ change the state of $M_1$ recorded in $M_2$'s control to complete the simulation of one move of $M_1$. If the new state of $M_1$ is accepting, then $M_2$ accepts.

## NUMERICAL PROBLEMS

***Prob.1. Design a turing machine to recognize all strings consisting of even number of 1's.*** ***(R.G.P.V., Dec. 2009)***

***Sol.*** The construction is made by defining moves in the following manner–

(i) $q_1$ is the initial state. $M$ enters state $q_2$ on scanning 1 and writes $b$.

(ii) If $M$ is in state $q_2$ and scans 1, it enters $q_1$ and writes $b$.

(iii) $q_1$ is the only accepting state.

So $M$ accepts a string if it exhausts all input symbols and finally in state $q_1$.

Symbolically,

$$M = (\{q_1, q_2\}, \{1, b\}, \{1, b\}, \delta, q_1, b, \{q_1\})$$

where $\delta$ is defined by table 5.3.

**Table 5.3 Transition Table**

| *Present State* | *1* |
|---|---|
| $\rightarrow (q_1)$ | $bq_2R$ |
| $q_2$ | $bq_1R$ |

Let us obtain the computation sequence of 11. Thus, $q_111 \vdash bq_21 \vdash bbq_1$. As $q_1$ is an accepting state, 11 is accepted. $q_1111 \vdash bq_211 \vdash bbq_11 \vdash bbbq_2$. $M$ halts and as $q_2$ is not an accepting state, 111 is not accepted by $M$.

***Prob.2. Construct TM accepting language***

$$L = \{w/w \text{ has even no. of } 2\}$$

***(R.G.P.V., June 2015)***

***Sol.*** Refer to Prob.1.

***Prob.3. Design a TM that accepts the language denoted by the regular expression $00^*$ for $\Sigma = \{0, 1\}$.***

***Sol.*** To solve the problem, we use the following logic –

(i) Change the state when first 0 comes, because regular expression contains atleast one 0.

(ii) The next state is final state for any number of 0.

(iii) Don't define any transition with input 1. So if at any time a 1 is encountered, the machine will halt without accepting the string. Symbolically

$$M = (\{q_0, q_1\}, \{0, 1\}, \{B\}, \delta, q_0, B, \{q_1\})$$

where $\delta$ is defined by table 5.4.



**Table 5.4**

| *Current State* | *Input* | | *B* |
|---|---|---|---|
| | *0* | *1* | |
| $\rightarrow q_0$ | $(q_1, B, R)$ | – | – |
| $(q_1)$ | $(q_1, B, R)$ | – | Accept |

***Prob.4. Design a turing machine that copies string of 1's.***

***Or***

***Find a machine that performs the following computation –***

$$q_0u \vdash^* q_fuu,$$

*for any* $u \in \{1\}^+$.

***Sol.*** To solve the problem, we implement the following intuitive process –

(i) Replace every 1 by an $x$

(ii) Find the rightmost $x$ and replace it with 1

(iii) Move to the right end of the current non-blank region and create a 1 there

(iv) Repeat steps (ii) and (iii) until there are no more $x$'s.

A turing machine version of this process is as follows –

$$\delta(q_0,1) = (q_1,x,R)$$
$$\delta(q_1,1) = (q_1,x,R)$$
$$\delta(q_1,B) = (q_2,B,L)$$
$$\delta(q_2,x) = (q_3,1,L)$$
$$\delta(q_2,1) = (q_2,1,L)$$
$$\delta(q_2,B) = (q_f,B,R)$$
$$\delta(q_3,1) = (q_3,1,R)$$
$$\delta(q_3,B) = (q_2,1,L)$$

where $q_f$ is the only final state. This may be somewhat hard to see at first. So let us trace the program with the simple string 11. The computation performed in this case is as follows –

$$\begin{aligned} q_011 &\vdash xq_11 \vdash xxq_1B \vdash xq_2x \vdash x1q_3B \\ &\vdash xq_211 \vdash q_2x11 \vdash 1q_311 \vdash 11q_31 \\ &\vdash 111q_3B \vdash 11q_211 \vdash 1q_2111 \vdash q_21111 \\ &\vdash q_2B1111 \vdash q_f1111 \end{aligned}$$

***Prob.5. M is a turing machine represented by the transition system ahead. Obtain the computation sequence of M for processing the input string 0011 –***

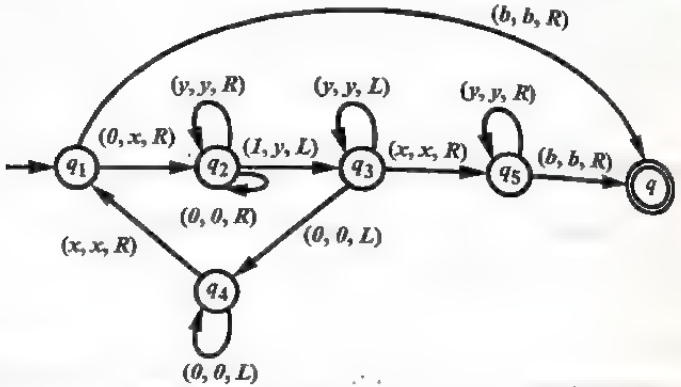



***Fig. 5.17 Transition System for M***

***(R.G.P.V., Dec. 2009)***

***Sol.*** The initial tape input is $b0011b$. Let us assume that $M$ is in state $q_1$ and the R/W head scans 0 (the first 0). We can represent this as in fig. 5.18.

$$\underset{q_1}{\overset{\downarrow}{b0011b}}$$

The figure can be represented by . From given fig. 5.17 we see that there is a directed edge from $q_1$ to $q_2$ with label $(0, x, R)$. So the current symbol 0 is replaced by $x$ and the head moves right.

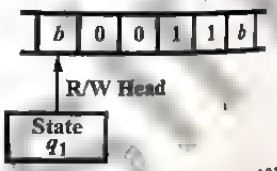



***Fig. 5.18 TM Processing 0011***

$$\underset{q_2}{\overset{\downarrow}{bx011b}}$$

The new state is $q_2$. Thus, we get .

The change brought about by processing the symbol 0 can be represented as –

$$\underset{q_1}{\overset{\downarrow}{b0011b}} \xrightarrow{(0,\,x,\,R)} \underset{q_2}{\overset{\downarrow}{bx011b}}$$

The entire computation sequence reads as follows –

$$\underset{q_1}{\overset{\downarrow}{b0011b}} \xrightarrow{(0,\,x,\,R)} \underset{q_2}{\overset{\downarrow}{bx011b}} \xrightarrow{(0,\,0,\,R)} \underset{q_2}{\overset{\downarrow}{bx011b}}$$

$$\xrightarrow{(1,\,y,\,L)} \underset{q_3}{\overset{\downarrow}{bx0y1b}} \xrightarrow{(0,\,0,\,L)} \underset{q_4}{\overset{\downarrow}{bx0y1b}} \xrightarrow{(x,\,x,\,R)} \underset{q_1}{\overset{\downarrow}{bx0y1b}}$$



$$\underset{q_2}{\overset{\downarrow}{bxxy1b}} \xrightarrow{(y,\,y,\,R)} \underset{q_2}{\overset{\downarrow}{bxxy1b}} \xrightarrow{(1,\,y,\,L)} \underset{q_3}{\overset{\downarrow}{bxxyyb}}$$

$$\xrightarrow{(y,\,y,\,L)} \underset{q_3}{\overset{\downarrow}{bxxyyb}} \xrightarrow{(x,\,x,\,R)} \underset{q_5}{\overset{\downarrow}{bxxyyb}} \xrightarrow{(y,\,y,\,R)} \underset{q_5}{\overset{\downarrow}{bxxyyb}}$$

$$\xrightarrow{(y,\,y,\,R)} \underset{q_5}{\overset{\downarrow}{bxxyyb}} \xrightarrow{(b,\,b,\,R)} \underset{q_6}{\overset{\downarrow}{bxxyybb}}$$

***Prob.6. Construct turing machine for accepting L –***

$$L = \{a^n b^n | n \geq 0\}$$ ***(R.G.P.V., June 2009)***

***Or***

***Construct a turing machine for a language having equal number of a's and b's in it over the input set $\Sigma = \{a, b\}$.*** ***(R.G.P.V., Dec. 2014)***

***Sol.*** The logic that we use for the turing machine to be constructed is as follows –

The turing machine will remember leftmost $a$, by replacing it with $B$, then it moves the tape head right keeping the symbols it scans as it is, until it gets rightmost $b$, it remembers rightmost $b$, by replacing it with $B$, and moves the tape head left keeping the symbols it scans as it is till it reaches the $B$, on getting $B$, it moves the tape head one position right and repeats the above cycle if it gets $a$. If it gets $B$ instead of $a$, then it is an indication of the fact the string is of the form $a^n b^n$, hence the turing machine enters into the final state. Therefore, the moves of the turing machine are given below in the form of transition table as shown in table 5.5.

**Table 5.5 Transition Table**

| | $a$ | $b$ | $B$ |
|---|---|---|---|
| $\rightarrow q_0$ | $(q_1, B, R)$ | | $(q_4, B, R)$ |
| $q_1$ | $(q_1, a, R)$ | $(q_1, b, R)$ | $(q_2, B, L)$ |
| $q_2$ | | $(q_3, B, R)$ | |
| $q_3$ | $(q_3, a, R)$ | $(q_3, b, R)$ | $(q_0, B, R)$ |
| $(q_4)$ | | | |

Therefore, the turing machine

$$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{a, b\}, \{a, b, B\}, \delta, q_0, B, \{q_4\})$$

where $\delta$ is given above.

The transition diagram corresponding to the above table is shown below –

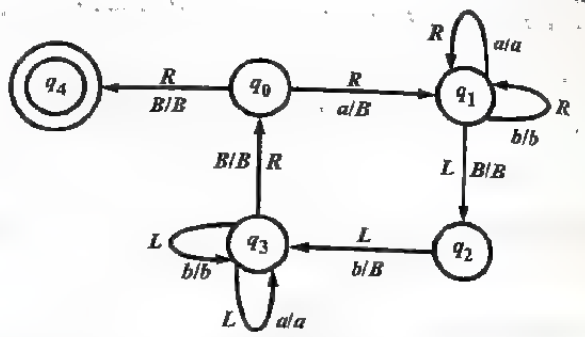



*Fig. 5.19*

***Prob.7. Design a turing machine M to recognize the language –***
***$\{1^n 2^n 3^n \mid n \geq 1\}$.*** ***(R.G.P.V., June 2006)***

***Or***

***Construct turing machine to accept the following language –***
***$L = \{a^n b^n c^n \mid n \geq 1\}$.*** ***(R.G.P.V., June 2008)***

***Or***

***Construct turing machine for the following set –***
***$L = \{a^n\ b^n\ c^n \mid n \geq 1\}$.*** ***(R.G.P.V., Dec. 2006)***

***Or***

***Design a turing machine to accept –***
***$L = \{a^n b^n c^n \mid n \geq 1\}$.*** ***(R.G.P.V., Dec. 2003)***

***Sol.*** Before designing the required turing machine $M$, let us evolve a procedure for processing the input string 112233. After processing, we require the ID to be of the form $BBBBBBq_7$.

The processing is done by using five steps –

**Step 1** – $q_1$ is the initial state. The R/W head scans the leftmost 1, replaces 1 by $B$, and moves to the right. $M$ enters $q_2$.

**Step 2** – On scanning the leftmost 2, the R/W head replaces 2 by $B$ and moves to the right. $M$ enters $q_3$.

**Step 3** – On scanning the leftmost 3, the R/W head replaces 3 by $B$, and moves to the right. $M$ enters $q_4$.

**Step 4** – After scanning the rightmost 3, the R/W head moves to the left until it finds the leftmost 1. As a result, the leftmost 1, 2, and 3 are replaced by $B$.

**Step 5** – Steps 1-4 are repeated until all 1's, 2's and 3's are replaced by blanks.



The change of ID's due to processing of 112233 is given as –

$$q_1112233 \vdash Bq_212233 \vdash B1q_22233 \vdash$$
$$B1Bq_3233 \vdash B1B2q_333 \vdash B1B2Bq_43 \vdash$$
$$B1B2q_5B3 \vdash B1Bq_52B3 \vdash B1q_5B2B3 \vdash$$
$$Bq_51B2B3 \vdash q_6B1B2B3 \vdash Bq_11B2B3 \vdash$$
$$BBq_2B2B3 \vdash BBBq_22B3 \vdash BBBBq_3B3 \vdash$$
$$BBBBBq_33 \vdash BBBBBBq_4B \vdash BBBBBq_7BB$$

Thus, $q_1112233 \vdash^* q_7BBBBBB$

As $q_7$ is an accepting state, the input string 112233 is accepted.

Now we can construct the transition table for $M$. It is given in table 5.6.

**Table 5.6 Transition Table**

| Present State | Input Tape Symbol | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | B |
| → $q_1$ | $BRq_2$ | | | $BRq_1$ |
| $q_2$ | $1Rq_2$ | $BRq_3$ | | $BRq_2$ |
| $q_3$ | | $2Rq_3$ | $BRq_4$ | $BRq_3$ |
| $q_4$ | | | $3Lq_5$ | $BLq_7$ |
| $q_5$ | $1Lq_6$ | $2Lq_5$ | | $BLq_5$ |
| $q_6$ | $1Lq_6$ | | | $BRq_1$ |
| (*$q_7$*) | | | | |

It can be seen from the table that strings other than those of the form $0^n1^n2^n$ are not accepted. It is advisable to compute the computation sequence for strings like 1223, 1123, 1233 and then see that these strings are rejected by $M$.

***Prob.8. Design a turing machine to recognize the following language –***
***$\{ww^R \mid w$ is in $(0 + 1)^*\}$*** ***(R.G.P.V., June 2004)***
***where $w^R$ is the reverse of w.***

***Or***

***Construct a turing machine that can accept the set of all even-length palindromes over {0, 1}.***

***Sol.*** Before designing the required turing machine $M$, let us evolve a procedure for processing the input string 1001. After processing, we require the ID to be of the form $BBBBq_7$.

We have the following steps for processing even-length palindromes –

**Step 1** – The turing machine $M$ scans the first symbol of the input tape (0, or 1), erase it and changes state ($q_1$ or $q_2$).

**Step 2** – $M$ scans the remaining part without changing the tape symbol until it encounters $B$.

**Step 3** – The R/W head moves to the left. If the rightmost symbol tallies with the leftmost symbol is erased. Otherwise $M$ halts.

**Step 4** – The R/W head moves to the left until $B$ is encountered.

**Step 5** – Steps 1-4 are repeated after changing the states suitably.

The changes of IDs due to processing of 1001 is given as –

$q_0 1001 \vdash Bq_2 001 \vdash^* B001q_2B \vdash B00q_4 1B \vdash B0q_6 0B \vdash^* q_6 B00B \vdash Bq_0 00B \vdash BBq_1 0B \vdash BB0q_1 B \vdash BBq_3 0B \vdash Bq_5 BBB \vdash BBq_0 BB \vdash^* BBBBq_7$.

Thus, $q_0 1001 \vdash^* BBBBq_7$

As $q_7$ is an accepting state, the input string 1001 is accepted.

Now we can construct the transition table $M$. The transition table is given in table 5.7.

**Table 5.7 Transition Table**

| *Present State* | *Input Tape Symbol* | | |
|---|---|---|---|
| | *0* | *1* | *B* |
| $\rightarrow q_0$ | $BRq_1$ | $BRq_2$ | $BRq_7$ |
| $q_1$ | $0Rq_1$ | $1Rq_1$ | $BLq_3$ |
| $q_2$ | $0Rq_2$ | $1Rq_2$ | $BLq_4$ |
| $q_3$ | $BLq_5$ | | |
| $q_4$ | | $BLq_6$ | |
| $q_5$ | $0Lq_5$ | $1Lq_5$ | $BRq_0$ |
| $q_6$ | $0Lq_6$ | $1Lq_6$ | $BRq_0$ |
| $(q_7)$ | | | |

It can be seen from the table that only the strings of the form $ww^R$ are accepted where $w \in (0+1)^*$. Hence $M$ recognizes the language

$L = \{ww^R | w \in (0+1)^*\}$

***Prob.9. Define turing machine. Draw a turing machine for the language $\{0^n 1^n 0^n \mid n \geq 1\}$.*** *(R.G.P.V., Dec. 2007)*

***Or***



***Design a turing machine to recognize the following language –***
***$L = \{0^n 1^n 0^n \mid n \geq 1\}$.***

*(R.G.P.V., June 2004, Dec. 2012)*

***Or***

***Construct turing machine for the following set –***
***$L = \{a^n b^n a^n \mid n \geq 1\}$.***

*(R.G.P.V., June 2007)*

***Sol.*** **Turing Machine** – Refer to Q.2.

We have the following steps for processing $0^n 1^n 0^n$ –

**Step 1** – $q_0$ is the initial state. The R/W head scans the leftmost 0, replaces 0 by $B$, and move to the right. $M$ enters $q_1$.

**Step 2** – On reading the leftmost 1, the R/W head replaces 1 by $B$ and moves to the right. $M$ enters $q_2$.

**Step 3** – On scanning the leftmost 0 after 1, the R/W head replaces 0 by $B$ and moves to the right. $M$ enters $q_4$.

**Step 4** – After reading the next 0, the R/W head moves to the left until it finds the leftmost 0. As a result, the leftmost 0, 1 and 0 are replaced by $B$.

**Step 5** – Steps 1-4 are repeated until all 0's, 1's and 0's are replaced by blanks.

Before constructing the transition table for the required turing machine $M$, let us see the sequence of moves for processing the input string 001100. The change of IDs in the processing of 001100 is given as

$q_0 001100 \vdash Bq_1 01100 \vdash B0q_1 1100 \vdash B0Bq_2 100 \vdash B0Bq_2 00 \vdash B0B1Bq_3 0 \vdash B0B1q_4 B0 \vdash^* Bq_4 0B1B0 \vdash q_5 B0B1B0 \vdash Bq_0 0B1B0 \vdash BBq_1 B1B0 \vdash BBBq_1 1B0 \vdash BBBBq_2 B0 \vdash BBBBBq_2 0 \vdash BBBBBBq_3 B \vdash BBBBBq_6 BB$

Thus, $q_0 001100 \vdash^* BBBBBBq_6$

As $q_6$ is an accepting state, hence the input string 001100 is accepted.

Now we construct the transition table for $M$. It is given in table 5.8.

**Table 5.8 Transition Table**

| *Present State* | *Input Tape Symbol* | | |
|---|---|---|---|
| | *0* | *1* | *B* |
| $\rightarrow q_0$ | $BRq_1$ | | $BRq_0$ |
| $q_1$ | $0Rq_1$ | $BRq_2$ | $BRq_1$ |
| $q_2$ | $BRq_3$ | $1Rq_2$ | $BRq_2$ |
| $q_3$ | $0Lq_4$ | | $BLq_6$ |
| $q_4$ | $0Lq_5$ | $1Lq_4$ | $BLq_4$ |
| $q_5$ | $0Lq_5$ | | $BRq_0$ |
| $(q_6)$ | | | |

It can be seen from the table that only the strings of the form of $0^n 1^n 0^n$ are accepted by $M$. Hence the turing machine $M$ recognize the language

$L = \{0^n 1^n 0^n | n \geq 1\}$

***Prob.10. Construct TM for following language L over alphabets a, b and c –***

$$L = \{a^n b^{2n} c^n \mid n \geq 0\}$$

***Sol.*** The construction of the required turing machine is given by the following steps –

**Step 1** – $q_0$ is the initial state. On scanning the leftmost $a$, the R/W head replaces $a$ by $B$, and moves to the right by changing state $q_0$ to $q_1$.

**Step 2** – On reading the leftmost $b$, the R/W head replaces two $b$'s by $B$'s and moves to the right by changing states $q_1$ to $q_2$ and $q_3$.

**Step 3** – On scanning the leftmost $c$, the R/W head replaces $c$ by $B$, and moves to the right by changing state $q_3$ to $q_4$.

**Step 4** – After reading the next $c$, the R/W head move to the left until it finds the rightmost $a$ and $M$ changes its state $q_4$ to $q_5$. After reading the rightmost $a$, the R/W head moves to the left without replacing $a$. When it reads $B$ on $q_5$ the R/W head moves to the right and $M$ enters $q_6$. As a result, the leftmost $a$, $b$, and $c$ are replaced by $B$.

**Step 5** – Steps 1-4 are repeated until all $a$'s, $b$'s and $c$'s are replaced by blanks.

Let us see the processing of input string *abbc*. The sequence of moves in processing of *abbc* is given as follows

$$q_0abbc \vdash Bq_1bbc \vdash BBq_2bc \vdash BBBq_3c \vdash BBBBq_4B \vdash BBBq_7BB$$

Thus, $q_0abbc \overset{*}{\vdash} BBBBq_7$

As $q_7$ is the final state, the input string *abbc* is accepted by TM.

Now we can construct the transition table for TM. It is given in table 5.9.

**Table 5.9 Transition Table**

| Present State | Input Tape Symbol | | | |
|---|---|---|---|---|
| | *a* | *b* | *c* | *B* |
| → $q_0$ | $BRq_1$ | | | $BRq_7$ |
| $q_1$ | $aRq_1$ | $BRq_2$ | | $BRq_1$ |
| $q_2$ | | $BRq_3$ | | |
| $q_3$ | | $bRq_3$ | $BRq_4$ | $BRq_3$ |
| $q_4$ | | | $cLq_5$ | $BLq_7$ |
| $q_5$ | $aLq_6$ | $bLq_5$ | | $BLq_5$ |
| $q_6$ | $aLq_6$ | | | $BRq_0$ |
| (→) $q_7$ | | | | |



It can be seen from the table that the string of the form of $a^n b^{2n} c^n$ are accepted by TM.

If $n = 0$, the TM should accept the blank symbol. From the first row of the table we can see that the TM also accepts the blank symbol. Hence the TM recognizes the language

$$L = \{a^n b^{2n} c^n \mid n \geq 0\}$$

***Prob.11. Construct TM for the following language L over the alphabets a, b and c –***

$$L = \{a^n b^{2n} c^{3n} \mid n \geq 1\}.$$

***Sol.*** The following steps give the construction of the required TM –

**Step 1** – $q_0$ is the initial state. On scanning the leftmost $a$, the R/W head replaces $a$ by $B$ and moves to the right. $M$ changes state $q_0$ to $q_1$.

**Step 2** – On scanning the leftmost $b$, the R/W head replaces two $b$'s by $B$'s and moves to the right. $M$ changes its states from $q_1$ to $q_2$ and $q_2$ to $q_3$ with replacing two consecutive $b$'s.

**Step 3** – On scanning the leftmost $c$, the R/W head replaces three $c$'s with $B$'s and moves to the right. $M$ changes its states $q_4$, $q_5$ and $q_6$.

**Step 4** – After scanning the next $c$, the R/W head returns to the leftmost $a$ without making any replacement. $M$ changes states $q_7$ and $q_8$. As a result, the leftmost one $a$, two $b$'s and three $c$'s are replaced by $B$.

**Step 5** – Steps 1-4 are repeated until all $a$'s, $b$'s and $c$'s are replaced by blanks.

**Table 5.10 Transition Table**

| Present State | Input Tape Symbol | | | |
|---|---|---|---|---|
| | *a* | *b* | *c* | *B* |
| → $q_0$ | $BRq_1$ | | | $BRq_0$ |
| $q_1$ | $aRq_1$ | $BRq_2$ | | $BRq_1$ |
| $q_2$ | | $BRq_3$ | | |
| $q_3$ | | $bRq_3$ | $BRq_4$ | $BRq_3$ |
| $q_4$ | | | $BRq_5$ | |
| $q_5$ | | | $BRq_6$ | $BRq_5$ |
| $q_6$ | | | $cLq_7$ | $BLq_9$ |
| $q_7$ | $aLq_8$ | $bLq_7$ | | $BLq_7$ |
| $q_8$ | $aLq_8$ | | | $BRq_0$ |
| (→) $q_9$ | | | | |

Let us see the sequence of moves in processing of the input string $abbccc$. The change of IDs due to processing of $abbccc$ is given as –

$$q_0abbccc \vdash Bq_1bbccc \vdash BBq_2bccc \vdash BBBq_3ccc \vdash BBBBq_4cc \vdash BBBBBq_5c \vdash BBBBBBq_6B \vdash BBBBBq_9BB$$

Thus, $q_0abbccc \overset{*}{\vdash} BBBBBBq_9$

As $q_9$ is an accepting state, the input string $abbccc$ is accepted by TM.

Now we construct the transition table for TM. It is given in table 5.10.

It can be seen from the table that the strings of the form of $a^nb^{2n}c^{3n}$ are accepted by the TM. In this problem $n \neq 0$ thus the TM does not accept the blank symbol initially. Hence the TM recognizes the language

$$L = \{a^nb^{2n}c^{3n} \mid n \geq 1\}$$

***Prob.12. Design turing machine for the language –***

$$L(G) = \{a^nb^ma^{n+m}/n \geq 1,\ m \geq 1\}.$$

***(R.G.P.V., Dec. 2010)***

***Sol.*** The required turing machine $T$ is defined as follows –

$$T = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where, $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_f\}$

$\Sigma = \{a, b\}$

$\Gamma = \{a, b, B\}$

$F = \{q_f\}$

and $\delta$ is given in transition table 5.11.

**Table 5.11**

| *States* | *a* | *b* | *B* |
|---|---|---|---|
| $\rightarrow q_0$ | $(q_1, B, R)$ | | |
| $q_1$ | $(q_1, a, R)$ | $(q_2, b, R)$ | |
| $q_2$ | $(q_3, a, R)$ | $(q_2, b, R)$ | |
| $q_3$ | $(q_3, a, R)$ | | $(q_4, B, L)$ |
| $q_4$ | $(q_5, B, L)$ | | |
| $q_5$ | $(q_5, a, L)$ | $(q_5, b, L)$ | $(q_6, B, R)$ |
| $q_6$ | $(q_1, B, R)$ | $(q_2, B, R)$ | $(q_f, B, R)$ |
| $(q_f)$ | Accept | Accept | Accept |

and transition diagram is shown in fig. 5.20.



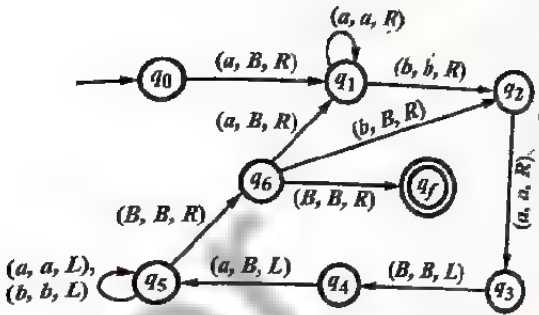



***Fig. 5.20 Transition Diagram for T***

***Prob.13. Design a TM that accepts the language L over the alphabets a and b.***

$$L = \{a^nb^n \mid n \geq 1\}$$

***Or***

***Construct a turing machine that recognizes the set $\{0^n1^n \mid n \geq 1\}$.***

***Or***

***Design a TM for language $\{L = a^nb^n \mid n \geq 1\}$.***

***(R.G.P.V., Dec. 2011, 2016)***

***Or***

***Design turing machine to accept –***

$$L = \{a^nb^n \mid n \geq 1\}.$$

***(R.G.P.V., June 2010)***

***Or***

***Design a turing machine for the following language –***

***The set of strings with an equal number of 0's and 1's.***

***(R.G.P.V., Dec. 2008)***

*Sol.* Starting at the leftmost $a$, we replace it by a symbol, say $x$. Then we let the R/W head move to right to find the leftmost $b$, which in turn replaced by a symbol, say $y$. After that we return to the leftmost $a$, replace it with an $x$, then move to the leftmost $b$ and replace it with $y$, and so on. Moving back and forth this way, we match each $a$ with a corresponding $b$. If after sometime no $a$'s or $b$'s remain, then the string must be in $L$.

Working out the details, we arrive at a complete solution for which

$$Q = \{q_0, q_1, q_2, q_3, q_4\},$$

$$F = \{q_4\},$$

$$\Sigma = \{a, b\},$$

$$\Gamma = \{a, b, x, y, B\}.$$

The transitions can be broken into several parts. The set of the follow. transition –

$$\delta(q_0, a) = (q_1, x, R)$$
$$\delta(q_1, a) = (q_1, a, R)$$
$$\delta(q_1, y) = (q_1, y, R)$$
$$\delta(q_1, b) = (q_2, y, L)$$

replaces the leftmost $a$ with an $x$, then causes the R/W head to move right to get the first $b$ and replacing it with a $y$. When $y$ is written, the machine enters a state $q_2$, indicating that an $a$ has been successfully paired with a $b$.

The next set of transitions reverse the direction until an $x$ is encountered, repositions the R/W head over the leftmost $a$, and returns control to the initial state.

$$\delta(q_2, y) = (q_2, y, L),$$
$$\delta(q_2, a) = (q_2, a, L),$$
$$\delta(q_2, x) = (q_0, x, R).$$

We are now back in the initial state $q_0$, ready to deal with the next $a$ and $b$.

After one pass through this part of the computation, the machine will have carried out the partial computation.

$$q_0aa \ldots abb \ldots b \overset{*}{\vdash} xq_0a \ldots ayb \ldots b,$$

so that a single $a$ has been matched with a single $b$. After two passes, we will have completed the partial computation.

$$q_0aa \ldots abb \ldots b \overset{*}{\vdash} xxq_0 \ldots ayy \ldots b.$$

and so on indicating that the process of machine is being carried out properly.

When the input is a string $a^nb^n$, the rewriting continuous this way, halting only when there are no more $a$'s and $b$'s to be erased. When looking for the leftmost $a$, the R/W head moves to the left with the machine in state $q_2$. When an $x$ is found, the direction is reversed to get the $a$. But now, instead of finding an $a$ it will find a $y$. To terminate, a final check is made to see if all $a$'s and $b$'s have been replaced (to detect input where an $a$ follows a $b$). This can be done by the following transitions

$$\delta(q_0, y) = (q_3, y, R)$$
$$\delta(q_3, y) = (q_3, y, R)$$
$$\delta(q_3, B) = (q_4, B, R)$$

If we input a string not in the language the computation will stop in some non-final state. For example, if we give the machine a string $a^nb^m$, with $n > m$, the machine will eventually find a blank in state $q_1$. It will halt because no transition is specified for this case. Other input not in the language, will lead to a non-final halting state.



The input $aabb$ gives the following successive instantaneous descriptions –

$$\begin{aligned} q_0aabb &\vdash xq_1abb \vdash xaq_1bb \vdash xq_2ayb \\ &\vdash q_2xayb \vdash xq_0ayb \vdash xxq_1yb \\ &\vdash xxyq_1b \vdash xxq_2yy \vdash xq_2xyy \\ &\vdash xxq_0yy \vdash xxyq_3y \vdash xxyyq_3B \\ &\vdash xxyyBq_4B. \end{aligned}$$

At this point the turing machine halts in a final state, so the string $aabb$ is accepted.

***Prob.14. Present a turing machine that inserts symbol # in the beginning of a string on the turing tape. Assume $\Sigma = \{a, b\}$.*** **(R.G.P.V., Dec. 2015)**

***Sol.*** The required turing machine $T$ is defined as follows –

$$T = (Q, \Sigma, \Gamma, \delta, q, B, F)$$

where $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_f\}$
$\Sigma = \{a, b\}$
$\Gamma = \{\#, a, b, B\}$
$F = \{q_f\}$

and $\delta$ is given in transition table 5.12.

**Table 5.12 $\delta$ Transition for $T$**

| States | a | b | B |
|---|---|---|---|
| $\rightarrow q_0$ | $(q_0, a, R)$ | $(q_0, b, R)$ | $(q_1, B, L)$ |
| $q_1$ | $(q_2, B, R)$ | $(q_3, B, R)$ | $(q_5, B, R)$ |
| $q_2$ | | | $(q_4, a, L)$ |
| $q_3$ | | | $(q_4, b, L)$ |
| $q_4$ | | | $(q_1, B, L)$ |
| $q_5$ | | | $(q_f, \#, R)$ |
| $(q_f)$ | Halt | | |

and transition diagram is shown in fig. 5.21.

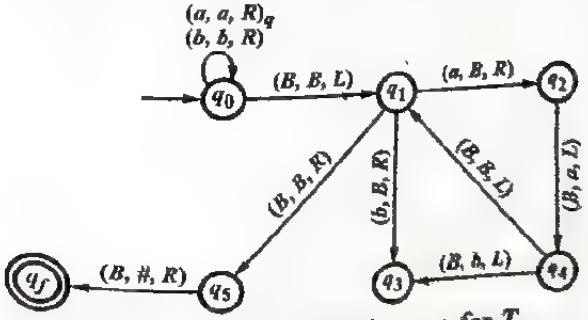
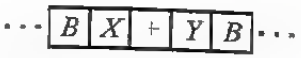



*Fig. 5.21 Transition Diagram for T*

***Prob.15. Design a turing machine that adds two numbers presented in binary notation and leaves the answer on the tape in binary form.*** ***(R.G.P.V., Dec. 2015)***

***Sol.*** Let as assume that binary numbers are stored on tape in following format –




where, $X$ and $Y$ can be any binary number
and number of bits $(X) \geq$ number of bits $(Y)$.

For example,

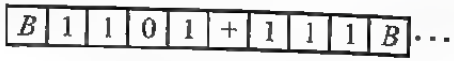



**Table 5.13**

| *States* | *0* | *1* | + | *Y* | *B* |
|---|---|---|---|---|---|
| $\rightarrow q_0$ | $(q_0, 0, R)$ | $(q_0, 1, R)$ | $(q_1, +, L)$ | | $(q_{12}, B, L)$ |
| $q_1$ | $(q_2, +, R)$ | $(q_3, +, R)$ | | | $(q_{11}, B, R)$ |
| $q_2$ | $(q_2, 0, R)$ | $(q_2, 1, R)$ | $(q_2, 0, R)$ | $(q_5, B, L)$ | $(q_4, B, L)$ |
| $q_3$ | $(q_3, 0, R)$ | $(q_3, 1, R)$ | $(q_3, 0, R)$ | $(q_6, 0, L)$ | $(q_5, B, L)$ |
| $q_4$ | $(q_7, B, R)$ | $(q_8, B, R)$ | | | |
| $q_5$ | $(q_8, B, R)$ | $(q_7, Y, R)$ | | | |
| $q_6$ | $(q_7, Y, R)$ | $(q_8, Y, R)$ | | | |
| $q_7$ | | | | | $(q_9, 0, L)$ |
| $q_8$ | | | | | $(q_9, 1, L)$ |
| $q_9$ | | | | $(q_{10}, Y, L)$ | $(q_{10}, B, L)$ |
| $q_{10}$ | $(q_{10}, 0, L)$ | $(q_{10}, 1, L)$ | $(q_1, +, L)$ | | |
| $q_{11}$ | $(q_{11}, B, R)$ | | $(q_{11}, B, R)$ | $(q_{12}, 1, R)$ | $(q_{12}, B, R)$ |
| $(q_{12})$ | Halt | | | | |

Then the turing machine which adds these binary numbers and leaves the answer on the tape in binary form is defined as –

$$T = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where, $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}, q_{11}, q_{12}\}$

$\Sigma = \{0, 1, +\}$

$\Gamma = \{0, 1, +, B, Y\}$

$F = \{q_{12}\}$

and $\delta$ transitions are defined in transition table 5.13.



The transition diagram of the resultant Turing machine is shown in fig. 5.22.

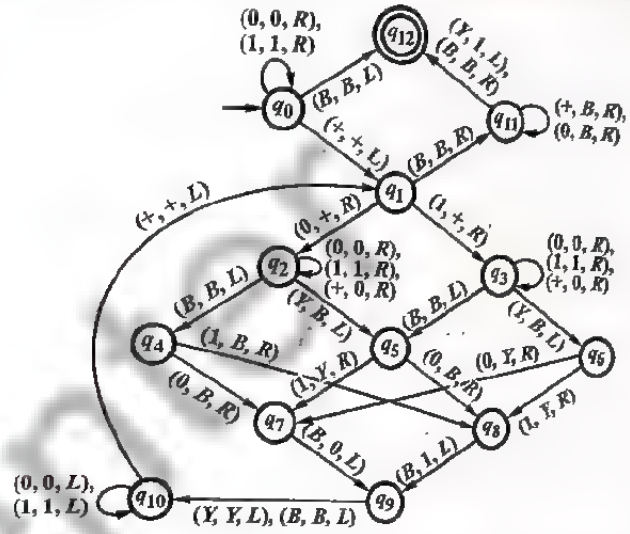



***Fig. 5.22***

***Prob.16. Construct a turing machine for checking the palindrome of a string of odd palindrome for $\Sigma = \{0, 1\}$.*** ***(R.G.P.V., June 2015)***

***Sol.*** Turing machine $T$ for checking the palindrome of a string of odd palindrome over $\Sigma = \{0, 1\}$ is given by –

$$T = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where, $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_f\}$

$\Sigma = \{0, 1\}$

$\Gamma = \{0, 1, B\}$

$F = \{q_f\}$

and $\delta$ is given by in transition table 5.14.

**Table 5.14 Turing Machine for Odd Palindrome**

| *States* | *0* | *1* | *B* |
|---|---|---|---|
| $\rightarrow q_0$ | $(q_1, B, R)$ | $(q_4, B, R)$ | |
| $q_1$ | $(q_1, 0, R)$ | $(q_1, 1, R)$ | $(q_2, B, L)$ |
| $q_2$ | $(q_3, B, L)$ | | $(q_f, B, R)$ |
| $q_3$ | $(q_3, 0, L)$ | $(q_3, 1, L)$ | $(q_0, B, R)$ |
| $q_4$ | $(q_4, 0, R)$ | $(q_4, 1, R)$ | $(q_5, B, L)$ |
| $q_5$ | | $(q_3, B, L)$ | $(q_f, B, R)$ |
| $(q_f)$ | Accept | Accept | Accept |

and transition diagram is shown in fig. 5.23.

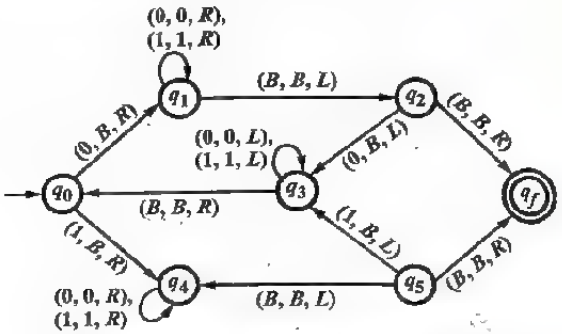



*Fig. 5.23 Transition Diagram for T.*

***Prob.17. Build a turing machine that accepts the language –***

$L = \{a^n b^{2n}\}$

***(R.G.P.V., Dec. 2013)***

***Or***

***Construct a turing machine that accepts the language $L = \{a^n b^{2n}\}$.***

***(R.G.P.V., Nov. 2018)***

***Sol.*** The resultant turing machine $T$ is defined as follows –

$T = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

where, $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_f\}$

$\Sigma = \{a, b\}$

$\Gamma = \{a, b, B\}$

$F = \{q_f\}$

and $\delta$ is shown in transition table 5.15.

**Table 5.15 Transition Table for *T***

| *States* | *a* | *b* | *B* |
|---|---|---|---|
| $\rightarrow q_0$ | $(q_1, B, R)$ | | $(q_f, B, R)$ |
| $q_1$ | $(q_1, a, R)$ | $(q_2, b, R)$ | |
| $q_2$ | | $(q_2, b, R)$ | $(q_3, B, L)$ |
| $q_3$ | | $(q_4, B, L)$ | |
| $q_4$ | | $(q_5, B, L)$ | |
| $q_5$ | $(q_5, a, L)$ | $(q_5, B, L)$ | $(q_0, B, R)$ |
| $(q_f)$ | Accept | Accept | Accept |

and transition diagram is shown in fig. 5.24.



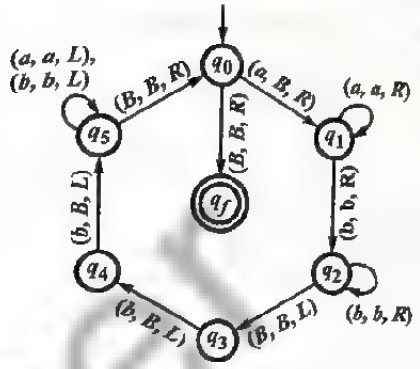



*Fig. 5.24 Transition Diagram for T*

***Prob.18. Construct a TM for language of even no. of 1's and even no. of 0's over $\Sigma = \{0, 1\}$.*** ***(R.G.P.V., Dec. 2011)***

***Sol.*** The required turing machine $T$ is defined as follows –

$T = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

where, $Q = \{q_0, q_1, q_2, q_3, q_f\}$

$\Sigma = \{0, 1\}$

$\Gamma = \{0, 1, B\}$

$F = \{q_f\}$

and $\delta$ transitions are shown in transition table 5.16.

**Table 5.16 Transition table for T**

| *States* | *0* | *1* | *B* |
|---|---|---|---|
| $\rightarrow q_0$ | $(q_1, 0, R)$ | $(q_3, 1, R)$ | $(q_f, B, L)$ |
| $q_1$ | $(q_0, 0, R)$ | $(q_2, 1, R)$ | |
| $q_2$ | $(q_3, 0, R)$ | $(q_1, 1, R)$ | |
| $q_3$ | $(q_2, 0, R)$ | $(q_0, 1, R)$ | |
| $(q_f)$ | Accept | Accept | Accept |

Transition diagram for turing machine $T$ is shown in fig. 5.25.

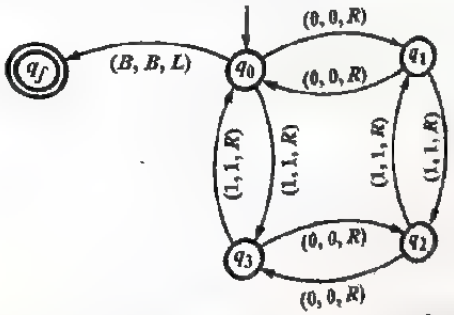



*Fig. 5.25 Transition Diagram for Turing Machine T*

***Prob.19. Construct a turing machine to compute the function $f(w) = w^R$, where $w \in \{0, 1\}^+$.*** ***(R.G.P.V., Dec. 2004)***

**Sol.** The turing machine $M$, which computes the function $F(w) = w^R$ for $w \in \{0, 1\}^+$ is defined as

$$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}, \{0, 1, X, B\}, \delta, q_0, B, \{q_6\})$$

where $\delta$ is defined in transition table 5.17.

**Table 5.17 Transition Table**

| Present State | Input Tape Symbol | | | |
|---|---|---|---|---|
| | 0 | 1 | X | B |
| $\rightarrow q_0$ | $(q_0,0,R)$ | $(q_0,1,R)$ | | $(q_1,B,L)$ |
| $q_1$ | $(q_2,X,R)$ | $(q_4,X,R)$ | $(q_1,X,L)$ | $(q_5,B,R)$ |
| $q_2$ | $(q_2,0,R)$ | $(q_2,1,R)$ | $(q_2,X,R)$ | $(q_3,0,L)$ |
| $q_3$ | $(q_3,0,L)$ | $(q_3,1,L)$ | $(q_1,X,L)$ | |
| $q_4$ | $(q_4,0,R)$ | $(q_4,1,R)$ | $(q_4,X,R)$ | $(q_3,1,L)$ |
| $q_5$ | $(q_6,0,R)$ | $(q_6,1,R)$ | $(q_5,B,R)$ | |
| $q_6$ | Halt | | | |

***Prob.20. Design a turing machine to compute the function $f(m, n) = m + n$ where m and n are non-negative numbers.*** ***(R.G.P.V., Dec. 2013)***

***Sol.*** Assume that $m$ and $n$ are positive integers in unary representation.

The computation of $f(m, n)$ can be visualized at a high level by means of the diagram in fig. 5.26. The diagram depicts that we first use a comparing machine to determine whether or not $m \geq n$. If so, the comparer sends a start signal to the adder, which then computes $m + n$. If not, an erasing program is started that changes every 1 to a blank.

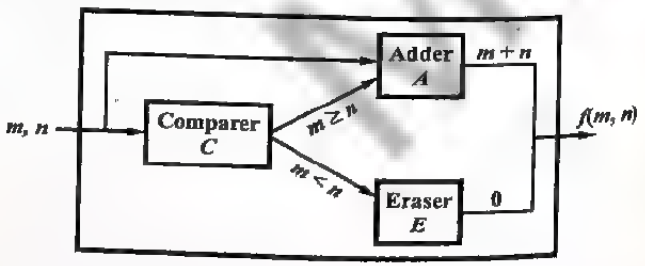



***Fig. 5.26***

The program for the comparer $C$ is written using a Turing machine having states indexed with $C$. For the adder, we use states indexed with $A$. For the eraser $E$, we construct a Turing machine having states indexed with $E$. The computations to be done by $C$ are –

$$q_{C,0}w(m)\ 0w(n) \vdash^* q_{A,0}w(m)\ 0w(n), \text{ if } m \geq n$$

and

$$q_{C,0}w(m)\ 0w(n) \vdash^* q_{E,0}w(m)\ 0w(n), \text{ if } m < n$$

If $q_{A,0}$ and $q_{E,0}$ are the initial states of $A$ and $E$, respectively, we see that $C$ starts either $A$ or $E$.

The computations performed by the adder will be

$$q_{A,0}w(m)\ 0w(n) \vdash^* q_{A,f}w(m+n)\ 0,$$

and that of the eraser $E$ will be

$$q_{E,0}w(m)\ 0w(n) \vdash^* q_{E,f}0$$

The result is a single turing machine that combines the action of $C$, $A$ and $E$ as shown in fig. 5.26.

***Prob.21. Proper subtraction $m \dot{-} n$ is defined to be $m - n$ for $m \geq n$ and zero $m < n$. Construct TM for this subtraction.***

***Or***

***Design a turing machine that can compute proper subtraction i.e. ...m_n where m and n are positive integers –***

$$m\_n = m - n \quad \text{if } m > n$$
$$m\_n = 0 \quad \text{if } m <= n$$

***(R.G.P.V., Dec. 2010)***

***Or***

***Design a turing machine that computes a function $f(m, n) = m \dot{-} n$ i.e., proper subtraction of two integers defined as,***

$$m \dot{-} n = \begin{cases} m - n & \text{if } m > n \\ 0 & \text{otherwise} \end{cases}$$

***(R.G.P.V., June 2009)***

***Sol.*** Let TM

$$M = (\{q_0, q_1, \ldots\ldots, q_6\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \{q_0\})$$

defined below started with $0^m10^n$ on its tape, halts with $0^{m \dot{-} n}$ on its tape. $M$ repeatedly replaces its leading 0 by blank, then searches right for a 1 followed by a 0 and changes the 0 to 1. Next, $M$ moves left until it encounters a blank and then repeats the cycle. The repetition ends if,

(i) Searching right for a 0, $M$ encounters a blank. Then, the $n$ 0's in $0^m10^n$ have all been changed to 1's and $n + 1$ of the $m$ 0's have been changed to $B$. $M$ replaces the $(n + 1)$ 1's by a 0 and $n$ $B$'s, leaving $(m - n)$ 0's on its tape.

(ii) Beginning the cycle, $M$ cannot find a 0 to change to a blank, because the first $m$ 0's already have been changed. Then $n \geq m$, so $m \dot{-} n = 0$. $M$ replaces all remaining 1's and 0's by $B$.

The function $\delta$ is described below –

(a) $\delta(q_0, 0) = (q_1, B, R)$

Begin the cycle. Replace the leading 0 by $B$.

(b) $\delta(q_1, 0) = (q_1, 0, R)$

$\delta(q_1, 1) = (q_2, 1, R)$

Search right, looking for the first (1).

(c) $\delta(q_2, 1) = (q_2, 1, R)$

$\delta(q_2, 0) = (q_3, 1, L)$

Search right pass 1's until encountering a 0. Change that 0 to 1.

(d) $\delta(q_3, 0) = (q_3, 0, L)$

$\delta(q_3, 1) = (q_3, 1, L)$

$\delta(q_3, B) = (q_0, B, R)$

Move left to a blank. Enter state $q_0$ to repeat the cycle.

(e) $\delta(q_2, B) = (q_4, B, L)$

$\delta(q_4, 1) = (q_4, B, L)$

$\delta(q_4, 0) = (q_4, 0, L)$

$\delta(q_4, B) = (q_6, 0, R)$

If in state $q_2$, a $B$ is encountered before a 0, we have situation (i) described above. Enter state $q_4$ and move left, changing all 1's to $B$'s until encountering a $B$. This $B$ is changed back to a 0, state $q_6$ is entered, and $M$ halts.

(f) $\delta(q_0, 1) = (q_5, B, R)$

$\delta(q_5, 0) = (q_5, B, R)$

$\delta(q_5, 1) = (q_5, B, R)$

$\delta(q_5, B) = (q_6, B, R)$

If in state $q_0$ a 1 is encountered instead of a 0, the first block of 0's has been exhausted, as in situation (ii) above. $M$ enters state $q_5$ to erase the rest of the tape, then enters $q_6$ and halts.

A sample computation of $M$ on input 0010 is –

$q_0 0010 \vdash Bq_1 010 \vdash B0s_1 10 \vdash B01q_2 0 \vdash$
$B0q_3 11 \vdash Bq_3 011 \vdash q_3 B011 \vdash Bq_0 011 \vdash$
$BBq_1 11 \vdash BB1q_2 1 \vdash BB11q_2 \vdash BB1q_4 1 \vdash$
$BBq_4 1 \vdash Bq_4 \vdash B0q_6$

On input 0100, $M$ behaves as follows –

$q_0 0100 \vdash Bq_1 100 \vdash B1q_2 00 \vdash Bq_3 110 \vdash$
$q_3 B110 \vdash Bq_0 110 \vdash BBq_5 10 \vdash BBBq_5 0 \vdash$
$BBBBq_5 \vdash BBBBBq_6$



***Prob.22. Design turing machine to add two numbers a and b.*** ***(R.G.P.V., June 2016)***

***Sol.*** We first have to choose some convention for representing positive integers. We will use unary notation, for simplicity, in which any positive integer $a$ is represented by

$$u(a) \in \{1\}^+ \text{ such that}$$

$$|u(a)| = a$$

We must also decide how $a$ and $b$ are placed on the tape initially and how their sum is to appear at the end of the computation. We will assume that $u(a)$ and $u(b)$ are on the tape in unary notation, separated by a single 0, with R/W head on the leftmost symbol of $u(a)$.

After the computation, $u(a+b)$ will be on the tape followed by a single 0, and the R/W head will be positioned at the left end of the result. Thus, we want to design a turing machine for performing the following computation –

$$q_0 u(a)\, 0 u(b) \overset{*}{\vdash} q_f\, u(a+b)\, 0,$$

where $q_f$ is a final state. Constructing a program for this is relatively simple. All we need to do is to move the separating 0 to the right end of $u(y)$, so that the addition amounts to nothing more than the coalescing of the two strings. To achieve this, we construct $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, with

$$Q = \{q_0, q_1, q_2, q_3, q_4\},$$

$$F = \{q_4\}$$

$$\delta(q_0, 1) = (q_0, 1, R),$$

$$\delta(q_0, 0) = (q_1, 1, R),$$

$$\delta(q_1, 1) = (q_1, 1, R),$$

$$\delta(q_1, B) = (q_2, B, L),$$

$$\delta(q_2, 1) = (q_3, 0, L),$$

$$\delta(q_3, 1) = (q_3, 1, L),$$

$$\delta(q_3, B) = (q_4, B, R),$$

It is noted that in moving the 0 right we temporarily create an extra 1, a fact that is remembered by putting the machine into state $q_3$. The transition $\delta(q_2, 1) = (q_3, 0, R)$ is needed to remove this at the end of the computation. This can be seen from the sequence of instantaneous descriptions for adding 111 to 11.

$q_0 11101 1 \vdash 1q_0 11011 \vdash 11q_0 1011 \vdash 111q_0 011 \vdash 1111q_1 11$
$\vdash 11111q_1 1 + 111111q_1 B \vdash 11111q_2 1 \vdash 1111q_3 10 \overset{*}{\vdash} q_3 B11110 \vdash q_4 11110.$

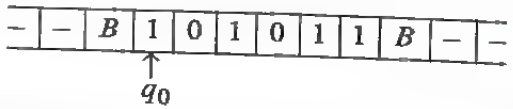
Adding numbers is one of the fundamental operations of any computer, one that plays a part in the synthesis of more complicated instructions. Other basic operations are copying string's and simple comparisons. Also this can be done on a turing machine.

***Prob.23. All natural numbers are either even or odd. Construct a turing machine that computes the parity function for natural numbers, that is, that computes –***

$$F(n) = \begin{cases} 0, & \text{if } n \text{ is even} \\ 1, & \text{if } n \text{ is odd} \end{cases}$$

***(R.G.P.V., Dec. 2010)***

***Sol.*** The input string is represented on the turing machine's tape in the form of




this machine starts in state $q_0$ at the beginning of the sequence, the '$B$' is to tell the machine whether the sequence ends. The machine needs two states one for 'ODD' and one for 'EVEN' parity and it changes states whether it encounters a '1'. The finite state machine is represented by table

| $q_i$ | $s_j$ | $q_{ij}$ | $s_{ij}$ | $d_{ij}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | $B$ | $H$ | 0 | – |

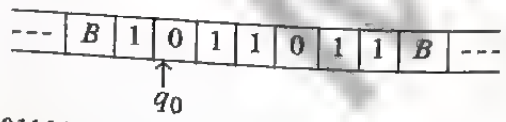
$q_0$

| $q_i$ | $s_j$ | $q_{ii}$ | $s_{ij}$ | $d_{ij}$ |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | $B$ | $H$ | 1 | – |

$q_1$

**Quintuples for Parity Counter**

In '$d_{ij}$', we are considering '0' for 'left' move and '1' for 'right' move. The operation of the machine is as follows that the machine ends up at the former site of the terminal '$B$' which it has replaced by the answer. The input sequence has been erased.




$$B\, q_0 1011011 B$$
$$\xrightarrow[q_1,0,R]{}$$
$$B\, 0\, q_1 0 1 1 0 1 1 B$$
$$\xrightarrow[q_1,0,R]{}$$



$$B\, 0\, 0\, q_1 1 1 0 1 1 B$$
$$\xrightarrow[q_0,0,R]{}$$
$$B\, 0\, 0\, q_0 1 0 1 1 B$$
$$\xrightarrow[q_1,0,R]{}$$
$$B\, 0\, 0\, 0\, 0\, q_1 0 1 1 B$$
$$\xrightarrow[q_1,0,R]{}$$
$$B\, 0\, 0\, 0\, 0\, q_1 1 1 B$$
$$\xrightarrow[q_0,0,R]{}$$
$$B\, 0\, 0\, 0\, 0\, 0\, q_1 1 B$$
$$\xrightarrow[q_1,0,R]{}$$
$$B\, 0\, 0\, 0\, 0\, 0\, 0\, 0\, q_1 B$$
$$B\, 0\, 0\, 0\, 0\, 0\, 0\, 0\, 1\, B$$

↳ this one is showing that given string containing odd parity.

***Prob.24. Design a turing machine M to recognize the language*** $\{ww : w \in \{a, b\}^*\}$. ***(R.G.P.V., Dec. 2010)***

***Sol.*** The idea behind the TM is to separate the processing into two parts– finding and marking the middle of the string and comparing the two halves. We accomplish the first task by working our way in from both ends simultaneously, changing symbols to their uppercase versions as we go. This means that our tape alphabet includes $A$ and $B$ in addition to the input symbols $a$ and $b$. Once we arrive at the middle – which happens only if the string is of even length – we may change the symbols in the first half back to their original form. The second part of the processing is to start at the beginning again and, for each lowercase symbol in the first half, compare it with the corresponding uppercase symbol in the second. We keep track of our progress by changing lowercase symbols to uppercase and erasing the matching uppercase symbols.

There are two ways that an input string can be rejected. If its length is odd, the TM will discover this in the first phase and crash. If it has even length but a symbol in the first half does not match the corresponding symbol in the second half, the TM will crash during the second phase.

The TM is shown in fig. 5.27. Again, we trace it for three strings – two that illustrate both ways the TM can crash and one that is in the language.

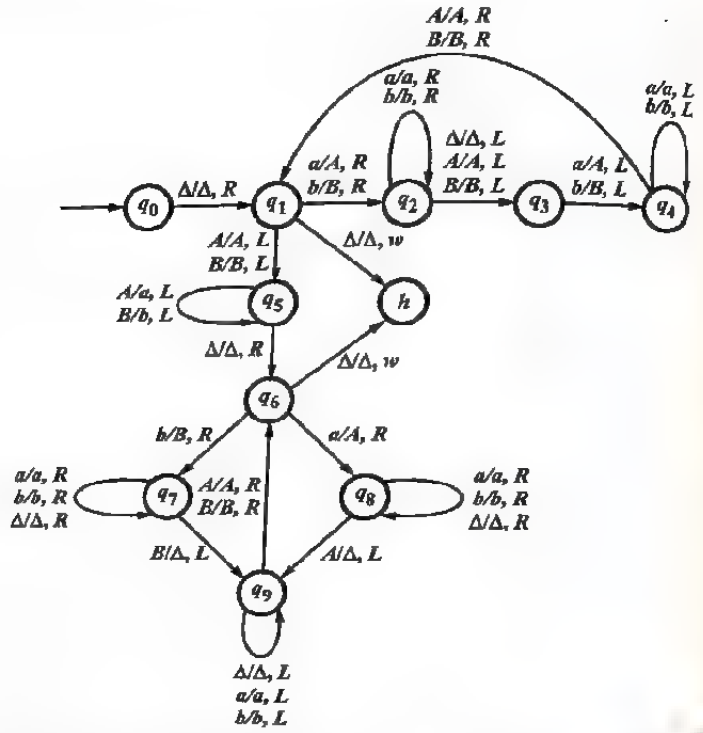



*Fig. 5.27 A Turing Machine to Accept {ww|w ∈ {a, b}*}*

$(q_0, \underline{\Delta}aba) \vdash (q_1, \Delta \underline{a}ba) \vdash (q_2, \Delta A\underline{b}a) \vdash^* (q_2, \Delta Aba\underline{\Delta})$
$\vdash (q_3, \Delta Ab\underline{a}) \vdash (q_4, \Delta A\underline{b}\,A) \vdash (q_4, \Delta \underline{A}b\,A)$
$\vdash (q_1, \Delta Ab\underline{a}) \vdash (q_2, \Delta AB\,\underline{A}) \vdash (q_3, \Delta A\underline{B}\,A)$
(crash)

$(q_0, \underline{\Delta}abaa) \vdash (q_1, \Delta \underline{a}baa) \vdash (q_2 \Delta A\underline{b}aa) \vdash^* (q_2, \Delta Abaa\underline{\Delta})$
$\vdash (q_3, \Delta Aba\underline{a}) \vdash (q_4, \Delta Ab\underline{a}\,A) \vdash^* (q_4, \Delta \underline{A}baA)$ @
$\vdash (q_1, \Delta A\underline{b}aA) \vdash (q_2, \Delta AB\underline{a}A) \vdash (q_2, \Delta ABa\underline{A})$
$\vdash (q_3, \Delta AB\underline{a}\,A) \vdash (q_4, \Delta A\underline{B}\,AA) \vdash (q_1, \Delta AB\underline{A}A)$
$\vdash (q_5, \Delta A\underline{B}AA) \vdash (q_5, \Delta \underline{A}bAA) \vdash (q_5, \underline{\Delta}abAA)$
(first phase completed)
$\vdash (q_6, \Delta \underline{a}bAA) \vdash (q_8, \Delta A\underline{b}\,AA) \vdash (q_8, \Delta Ab\underline{A}\,A)$
$\vdash (q_9, \Delta A\underline{b}\Delta A) \vdash (q_9, \Delta \underline{A}b\Delta A) \vdash (q_6, \Delta A\underline{b}\Delta A)$
$\vdash (q_7, \Delta AB\underline{\Delta}\,A) \vdash (q_7, \Delta A\,B\,\Delta\,\underline{A})$ (crash)

$(q_0, \underline{\Delta}abab) \vdash^*$

(same as previous case, up to third-from-last move)



$\vdash (q_6, \Delta A\underline{b}\Delta B) \vdash (q_7, \Delta AB\underline{\Delta}B) \vdash (q_7, \Delta AB\Delta\underline{B})$
$\vdash (q_9, \Delta AB\,\underline{\Delta}) \vdash (q_9, \Delta A\underline{B}) \vdash (q_6, \Delta AB\underline{\Delta})$
$\vdash (h, \Delta AB\,\underline{\Delta})$ (accept)

## RECURSIVE AND RECURSIVELY ENUMERABLE LANGUAGES, DECIDABLE AND UNDECIDABLE PROBLEMS – EXAMPLES, HALTING PROBLEM, REDUCIBILITY

***Q.27. Prove that the union of two recursively enumerable languages is recursively enumerable.***

***Ans.*** To prove the union of two recursively enumerable languages is recursively enumerable, let us consider $L_1$ and $L_2$ be two recursively enumerable langauges accepted by algorithms $M_1$ and $M_2$.

We construct $M$, which simulate simultaneously $M_1$ and $M_2$ on separate tapes. If either accepts, then $M$ accepts. The construction is shown in fig. 5.28.

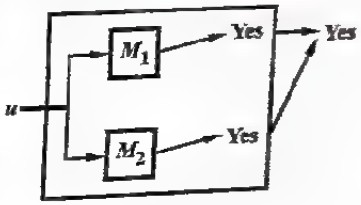



*Fig. 5.28*

***Q.28. Prove that the intersection of two recursively enumerable languages is also recursively enumerable.***

***Ans.*** Suppose that $T_1 = (Q_1, \Sigma, \Gamma_1, q_1, \delta_1)$ and $T_2 = (Q_2, \Sigma, \Gamma_2, q_2, \delta_2)$ are TMs accepting $L_1$ and $L_2$, respectively. We wish to construct TMs accepting $L_1 \cap L_2$.

The two-tape machine $T = (Q, \Sigma, \Gamma, q_0, \delta)$ begins by copying the input string $x$ onto tape 2 and inserting on $T_1$. Inserting the marker # at the beginning of each tape. The simultaneous simulation of tape 1 and $T_2$ on tape 2 is accomplished by allowing every possible move

$$\delta((p_1, p_2), (a_1, a_2)) = ((q_1, q_2), (b_1, b_2), (D_1, D_2))$$

where, for both values of $i$, $\delta_i(p_i, a_i) = (q_i, b_i, D_i)$. The possible outcomes of the simulation are –

(i) Neither $T_1$ nor $T_2$ ever stops, in which case $T$ never stops.

(ii) At least one of the two halts, in which case $T$ halts.

(iii) Machine can crash as soon as either $T_1$ or $T_2$ crashes, and it can halt only after both $T_1$ and $T_2$ have halted. Hence, we may conclude that $T$ accepts the language $L_1 \cap L_2$.

***Q.29. Give two properties of recursively enumerable set.***
***(R.G.P.V., Dec. 2016)***

***Ans.*** Refer to Q.27 and Q.28.

***Q.30. Explain the properties of recursive and recursive enumerable languages.***
***(R.G.P.V., Dec. 2011, 2012)***

***Or***

***Write a short note on recursive and recursively enumerable languages.***
***(R.G.P.V., June 2004)***

***Or***

***Discuss the properties of recursive and recursive enumerable languages.***
***(R.G.P.V., June 2015)***

Ans. **Recursively Enumerable Languages** – A language $L$ is said to be ***recursively enumerable,*** if there exists a turing machine that accepts $L$.

This definition implies only that there exists a turing machine $M$, such hat, for every $u \in L$.

$$q_0 u \overset{*}{\underset{M}{\vdash}} x_1 q_f x_2,$$

vith $q_f$ a final state. The definition does not say anything about what happens or $u$ not in $L$; it may be that the machine halts in a non-final state or that it ever halts and goes into an infinite loop.

**Recursive Languages** – A language $L$ on $\Sigma$ is said to be recursive, if iere exists a turing machine $M$ that accepts $L$ and that halts on every $u \in \Sigma^+$. t other words, we can say that a language is recursive iff there exists a embership algorithm for it or if there exists a TM that recognizes $L$.

If a langauge is recursive, then there exists an easily constructed $M$ umeration procedure. Suppose that $M$ is a turing machine that determines embership in a recursive language $L$. First, we construct another turing achine, say $M'$, that generates all strings in $\Sigma^+$, in proper order, say $u_1, u_2,$ As these strings are generated, they become the input to $M$, which is odified so that it writes strings on its tape only if they are in $L$.

There is also an enumeration procedure for every recursively enumerable gauge, but it is not easy to see. It is easy to see that every language for ich an enumeration procedure exists is recursively enumerable. We simply npare the given input string against successive strings generated by the umeration procedure. If $u \in L$, then we will eventually find a match, and the cess can be terminated.

***Q.31. Explain recursively enumerable language. Prove that every ursive language is recursively enumerable language. (R.G.P.V., June 2008)***



***Ans.*** **Recursively Enumerable Language** – Refer to Q.30.

***Proof*** – Every language that is recursive is recursively enumerable. If $T$ is a TM recognizing $L$, then a modified machine that crashes instead of leaving output 0 accepts $L$. If $T$ is a TM accepting $L$, there may strings not in $L$ for which $T$ crashes and therefore never produces an answer.

If a language is recursive, then there exists an enumeration procedure. Suppose that $M$ is a Turing machine that determines membership in a recursive language $L$. We first construct another turing machine, say $\hat{M}$, that generates all strings in $\Sigma^+$ in proper order, let us say $w_1, w_2, \ldots$. As these strings are generated, they become the input to $M$, which is modified so that it writes strings on its tape only if they are in $L$.

That there is also an enumeration procedure for every recursively enumerable language is not as easy to see. We cannot use the above argument as it stands, because if some $w_j$ is not in $L$, the machine $M$, when started with $w_j$ on its tape, may never halt and therefore never get to the strings in $L$ that follow $w_j$ in the enumeration: To make sure that this does not happen, the computation is performed in a different way. We first get $\hat{M}$ to generate $w_1$ and let $M$ execute one move on it. Then we let $\hat{M}$ generate $w_2$ and let $M$ execute one move on $w_2$, followed by the second move on $w_2$, the third step on $w_1$, and so on. From this it is clear that $M$ will never get into an infinite loop. Since any $w \in L$ is generated by $\hat{M}$ and accepted by $M$ in a finite number of steps, every string in $L$ is eventually produced by $M$.

It is easy to see that every language for which an enumeration procedure exists is recursively enumerable. We simply compare the given input string against successive strings generated by the enumeration procedure. If $w \in L$, we will eventually get a match, and the process can be terminated.

***Q.32. Explain the term recursively enumerable language.***
***(R.G.P.V., Dec. 2014)***

***Or***

***Explain recursively enumerable languages.*** ***(R.G.P.V., June 2009)***

***Or***

***Write short note on recursively enumerable languages.***
***(R.G.P.V., June 2003, Dec. 2005, 2007)***

***Ans.*** Refer to Q.30.

***Q.33. Prove that the complement of a recursive language is recursive.***

***Or***

***Prove that if $L$ is a recursive language so is $\overline{L}$.*** ***(R.G.P.V., Dec. 2008)***

***Q.29. Give two properties of recursively enumerable set.***

***Ans.*** Refer to Q.27 and Q.28. ***(R.G.P.V., Dec. 2016)***

***Q.30. Explain the properties of recursive and recursive enumerable languages.*** ***(R.G.P.V., Dec. 2011, 2012)***

***Or***

***Write a short note on recursive and recursively enumerable languages.*** ***(R.G.P.V., June 2004)***

***Or***

***Discuss the properties of recursive and recursive enumerable languages.*** ***(R.G.P.V., June 2015).***

Ans. **Recursively Enumerable Languages** – A language $L$ is said to be ***recursively enumerable,*** if there exists a turing machine that accepts $L$.

This definition implies only that there exists a turing machine $M$, such that, for every $u \in L$.

$$q_0 u \overset{*}{\underset{M}{\vdash}} x_1 q_f x_2,$$

with $q_f$ a final state. The definition does not say anything about what happens for $u$ not in $L$; it may be that the machine halts in a non-final state or that it never halts and goes into an infinite loop.

**Recursive Languages** – A language $L$ on $\Sigma$ is said to be recursive, if there exists a turing machine $M$ that accepts $L$ and that halts on every $u \in \Sigma^+$. In other words, we can say that a language is recursive iff there exists a membership algorithm for it or if there exists a TM that recognizes $L$.

If a langauge is recursive, then there exists an easily constructed $M$ enumeration procedure. Suppose that $M$ is a turing machine that determines membership in a recursive language $L$. First, we construct another turing machine, say $M$, that generates all strings in $\Sigma^+$, in proper order, say $u_1, u_2, \ldots$ As these strings are generated, they become the input to $M$, which is modified so that it writes strings on its tape only if they are in $L$.

There is also an enumeration procedure for every recursively enumerable langauge, but it is not easy to see. It is easy to see that every language for which an enumeration procedure exists is recursively enumerable. We simply compare the given input string against successive strings generated by the enumeration procedure. If $u \in L$, then we will eventually find a match, and the process can be terminated.

***Q.31. Explain recursively enumerable language. Prove that every recursive language is recursively enumerable language. (R.G.P.V., June 2008)***



***Ans.*** **Recursively Enumerable Language** – Refer to Q.30.

**Proof** – Every language that is recursive is recursively enumerable. If $T$ is a TM recognizing $L$, then a modified machine that crashes instead of leaving output 0 accepts $L$. If $T$ is a TM accepting $L$, there may strings not in $L$ for which $T$ crashes and therefore never produces an answer.

If a language is recursive, then there exists an enumeration procedure. Suppose that $M$ is a Turing machine that determines membership in a recursive language $L$. We first construct another turing machine, say $\hat{M}$, that generates all strings in $\Sigma^+$ in proper order, let us say $w_1, w_2, \ldots$. As these strings are generated, they become the input to $M$, which is modified so that it writes strings on its tape only if they are in $L$.

That there is also an enumeration procedure for every recursively enumerable language is not as easy to see. We cannot use the above argument as it stands, because if some $w_j$ is not in $L$, the machine $M$, when started with $w_j$ on its tape, may never halt and therefore never get to the strings in $L$ that follow $w_j$ in the enumeration: To make sure that this does not happen, the computation is performed in a different way. We first get $\hat{M}$ to generate $w_1$ and let $M$ execute one move on it. Then we let $\hat{M}$ generate $w_2$ and let $M$ execute one move on $w_2$, followed by the second move on $w_2$, the third step on $w_1$, and so on. From this it is clear that $M$ will never get into an infinite loop. Since any $w \in L$ is generated by $\hat{M}$ and accepted by $M$ in a finite number of steps, every string in $L$ is eventually produced by $M$.

It is easy to see that every language for which an enumeration procedure exists is recursively enumerable. We simply compare the given input string against successive strings generated by the enumeration procedure. If $w \in L$, we will eventually get a match, and the process can be terminated.

***Q.32. Explain the term recursively enumerable language.*** ***(R.G.P.V., Dec. 2014)***

***Or***

***Explain recursively enumerable languages.*** ***(R.G.P.V., June 2009)***

***Or***

***Write short note on recursively enumerable languages.*** ***(R.G.P.V., June 2003, Dec. 2005, 2007)***

***Ans.*** Refer to Q.30.

***Q.33. Prove that the complement of a recursive language is recursive.***

***Or***

***Prove that if L is a recursive language so is $\overline{L}$. (R.G.P.V., Dec. 2008)***

*Ans.* Let $L$ be a recursive language and $M$ be a turing machine that halts on all inputs and accepts $L$. Let us construct $M'$ from $M$ so that if $M$ enters a final state on input $u$, then $M'$ halts without accepting. If $M$ halts without accepting, $M'$ enters a final state. Since one of these two events occurs, $M'$ is an algorithm. Clearly $L(M')$ is the complement of $L$ and thus the complement of $L$ is a recursive language.

The construction of $M'$ from $M$ is given in fig. 5.29.

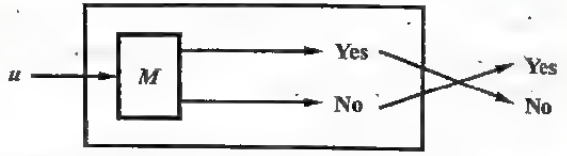



***Fig. 5.29 Construction Showing that Recursive Languages are Closed under Complementation***

***Q.34. What do you mean by recursive language ? Prove that complement of a recursive language is recursive.*** ***(R.G.P.V., Dec. 2013)***

*Ans.* Refer to Q.30 and Q.33.

***Q.35. Prove that the if a language L and its complement $\overline{L}$ are both recursively enumerable, then both languages are recursive. If L is recursive, the $\overline{L}$ is also recursive, and consequently both are recursively enumerable.***

*Ans.* If $L$ and $\overline{L}$ are both recursively enumerable, then there exist Turing Machines $M$ and $\hat{M}$ that serve as enumeration procedures for $L$ and $\overline{L}$, respectively. The first will produce $w_1$, $w_2$ ..... in $L$, the second $\hat{w}_1, \hat{w}_2$..... in $\overline{L}$. Suppose now we are given any $w \in \Sigma^+$. We first Let $M$ generate $w_1$ and compare it with $w$. If they are not the same, we let $\hat{M}$ generate $\hat{w}_1$ and compare again. If we need to continue, we next let $M$ generate $w_2$, then $\hat{M}$ generate $\hat{w}_2$, and so on. Any $w \in \Sigma^+$ will be generated either by $M$ or $\hat{M}$, so eventually we will get a match. If the matching string is produced by $M$, $w$ belongs to $L$, otherwise it is in $\overline{L}$. The process is a membership algorithm for both $L$ and $\overline{L}$, so they are both recursive.

For the converse, assume that $L$ is recursive. Then there exists a membership algorithm for it. But this becomes a membership algorithm for $\overline{L}$ by simply complementing its conclusion. Therefore $\overline{L}$ is recursive. Since any recursive language is recursively enumerable.

***Q.36. Prove that universal language is recursively enumerable.*** ***(R.G.P.V., Dec. 2007)***



*Ans.* We shall exhibit a three-tape TM $M_1$ accepting universal language $L_u$. The first tape of $M_1$ is the input tape, and the input head on that tape is used to look up moves of the TM $M$ when given code $<M, w>$ as input. Note that the moves of $M$ are found between the first two blocks of three 1's. The second tape of $M_1$ will simulate the tape of $M$.

The alphabet of $M$ is $\{0, 1, B\}$, so each symbol of $M$'s tape can be held in one tape cell of $M_1$'s second tape. Observe that if we did not restrict the alphabet of $M$, we would have to use many cells of $M_1$'s tape to simulate one of $M$'s cells, but the simulation could be carried out with a little more work. The third tape holds the state of $M$, with $q_i$ represented by $0^i$. The behaviour of $M_1$ is as follows –

(i) Check the format of tape 1 to see that it has a prefix and that there are no two codes that begin with $0^i10^j1$ for the same $i$ and $j$.

(ii) Initialize tape 2 to contain $w$, the portion of the input beyond the second block of three 1's. Initialize tape 3 to hold a single 0, representing $q_1$. All three tape heads are positioned on the leftmost symbols.

(iii) If tape 3 holds 00, the code for the final state, halt and accept.

(iv) Let $X_j$ be the symbol currently scanned by tape head 2 and let $0^i$ be the current contents of tape 3. Scan tape 1 from the left end to the second 111, looking for a substring beginning $110^i10^j1$. If no such string is found, halt and reject. $M$ has no next move and has not accepted. If such a code is found, let it be $0^i10^j10^k10^l10^m$. Then put $0^k$ on tape 3, print $X_l$ on the tape cell scanned by head 2 and move that head in direction $D_m$.

It is straightforward to check that $M_1$ accepts $<M, w>$ if and only if $M$ accepts $w$. It is also true that if $M$ runs forever on $w$, $M_1$ will run forever on $<M, w>$, and if $M$ halts on $w$ without accepting, $M_1$ does the same on $<M, w>$.

***Q.37. Prove that – (i) The union of two recursive languages is recursive. (ii) If a language L and its complement $\overline{L}$ are both recursively enumerable then L is recursive.*** ***(R.G.P.V., Dec. 2003)***

*Ans.* ***(i)*** To prove the union of two recursive languages is recursive, let us conisder $L_1$ and $L_2$ be two recursive languages accepted by algorithms $M_1$ and $M_2$. We construct $M$, which first simulates $M_1$. If $M_1$ accepts, then $M$ accepts. If $M_1$ rejects then $M$ simulates $M_2$ and accepts iff $M_2$ accpets. Since both $M_1$ and $M_2$ are algorithms, $M$ is guaranteed to halt. Clearly, $M$ accepts $L_1 \cup L_2$. The construction of $M$ is shown in fig. 5.30.

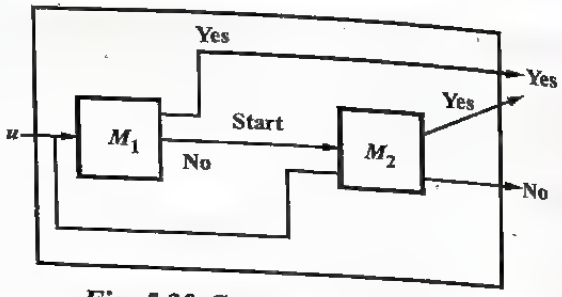



*Fig. 5.30 Construction of Union*

***(ii)*** To prove $L$ is recursive, if $L$ and its complement $\overline{L}$ both are recursively enumerable, let us consider $M_1$ and $M_2$ accept $L$ and $\overline{L}$, respectively.

We construct $M$, as given in fig. 5.31 to simulate simultaneously $M_1$ and $M_2$. $M$ accepts $u$ if $M_1$ accepts $u$ and rejects $u$ if $M_2$ accept $u$. Since $u$ is in either $L$ or $\overline{L}$, we know that exactly one of $M_1$ or $M_2$ will accept. Thus $M$ will always say either "yes" or "no", but will never say both.

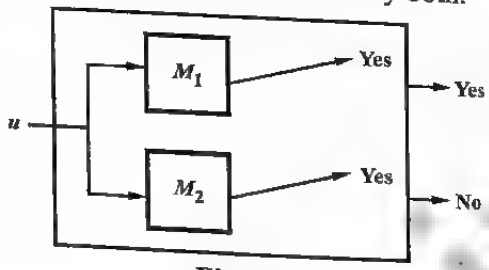



*Fig. 5.31*

It is noted that there is no priori limit on how long it may take before $M_1$ or $M_2$ accepts, but it is certain that one or the other will do so. Since $M$ is an algorithm that accepts $L$, it follows that $L$ is recursive.

***Q.38. Find the languages obtained from the following operations –***

***(i) Union of two recursive languages***

***(ii) Union of two recursively enumerable languages***

***(iii) If L and complement of L are recursively enumerable.***

***(R.G.P.V., Dec. 2014, June 2015)***

***Ans.*** Refer to Q.37 and Q.27.

***Q.39. Prove that, there exists a recursively enumerable language whose complement is not a recursively enumerable.***

***Ans.*** Suppose $\Sigma = \{x\}$, and consider the set of all turing machines with this input alphabet. This set is countable, so we can associate an order



$M_1, M_2, \ldots\ldots$with its elements. For each turing machine $M_i$, there is an associated recursively enumerable language $L(M_i)$. Conversely, for each recursively enumerable language on $\Sigma$, there is some turing machine that accepts it.

Now, we consider a new language $L$ defined as, for each $i \geq 1$, the string $x^i$ is in $L$ if and only if $x^i \in L(M_i)$. Clearly that the language $L$ is well defined, since the statement $x^i \in L(M_i)$, and hence $x^i \in L$, must either be true or false. Next, we consider the complement of $L$,

$$\overline{L} = \left\{x^i : x^i \notin L(M_i)\right\}, \quad \ldots\ldots(i)$$

which is also well defined but, as we will show, is not recursively enumerable.

Now, we will show this by contradiction, starting from the assumption that $\overline{L}$ is recursively enumerable, then there must be some turing machine, say $M_k$, such that –

$$\overline{L} = L(M_k) \quad \ldots(ii)$$

Consider the string $x^k$. Is it in $L$ or in $\overline{L}$? Let $x^k \in \overline{L}$. By equation (ii), this implies that –

$$x^k \in L(M_k)$$

But by equation (i) it implies that

$$x^k \notin \overline{L}$$

Conversely, if we assume that $x^k$ is in $L$, then $x^k \notin \overline{L}$ and equation (ii) implies that –

$$x^k \notin L(M_k)$$

But then from equation (i), we get

$$x^k \in \overline{L}$$

The contradiction is inescapable, and we must conclude that our assumption that $\overline{L}$ is recursively enumerable is false.

For showing the $L$ is recursively enumerable, we can use the known enumeration procedure for turing machines. Given $x^i$, we first find $i$ by counting the number of $x$'s. Then we use the enumeration procedure for turing machines to find $M_i$. Finally, we give its descriptions along with $x^i$ to a universal turing machine $M_u$ that simulates the action of $M$ on $x^i$. If $x^i$ is in $L$, the computation carried out by $M_u$ will eventually halt. The combined effects of this is a turing machine that accepts every $x^i \in L$. Therefore, by definition, $L$ is recursively enumerable.

***Q.40. What are decidable and undecidable problems ?*** ***(R.G.P.V., Dec. 2016)***

***Ans.*** **Decidable Problems** – Decidable problems are these problems for which there exist an algorithm for that problem where the result of computation is "Yes" or "No". For example, let us consider a statement "A context-free grammar is in Chomsky Normal Form." For some context-free grammar this is false, for other it is true. If there exists any algorithm which decides the given statement is true or false for any problem. Then the problem is decidable problem.

**Undecidable Problems** – A problem is undecidable if there is no algorithm which will correctly give a solution to given problem in form of "Yes" or "No".

Examples of some undecidable problems are

(i) The Halting problem of turing machine,

(ii) Post correspondence problem.

***Q.41. Describe decidable and undecidable problem. Explain halting problem.*** ***(R.G.P.V., Dec. 2017)***

***Ans.*** **Decidable Problem** – Refer to Q.40.

**Undecidable Problem** – Refer to Q.40.

**Halting Problem** – For any turing machine, there are two cases arises -

(i) The turing machine will halt after some finite number of steps.

(ii) The turing machine will never halt.

The problem is, "Is it possible to determine for any given input whether the turing machine will ever halt or not ?"

This problem is known as halting problem of turing machine.

Simply in formal way –

"For any turing machine T and a string $w$, is $w \in L(T)$?" is halting problem of turing machine.

***Q.42. Explain process of reducibility.*** ***(R.G.P.V., June 2016)***

***Ans.*** In reducibility process, a problem is converted to another problem in such a way that a solution to second problem can be used to solve the first problem. Such reducibilities come up in everyday life even if we do not usually refer to them in this way. As an example, consider you want to find our way around a new city. It is easy to do if you had a map. Thus you can reduce the problem of finding your way around the city to the problem of obtaining a map. There are always two problems in reducibility, which we call A and B. If A reduces to B, we can use a solution to B to solve A. Reducibility says nothing about solving A and B alone, but only about the solvability of A in the presence of a solution to B. Reducibility also occurs in mathematical problems. Reducibility plays an important role in classifying problems by decidability and later in complexity theory.

## INTRODUCTION OF P, NP, NP COMPLETE, NP HARD PROBLEMS AND EXAMPLES OF THESE PROBLEMS

***Q.43. What is tractable and intractable problem ?*** ***(R.G.P.V., Dec. 2014)***

***Or***

***What is tractable and untractable problem ?*** ***(R.G.P.V., June 2015)***

***Or***

***Discuss tractable and intractable problem.*** ***(R.G.P.V., Dec. 2015)***

***Or***

***Differentiate between tractable and untractable problems.*** ***(R.G.P.V., Dec. 2016)***

***Ans.*** Algorithm whose execution time can be determined by a polynomial on the size of input or can be bounded by such a polynomial are called polynomial time algorithm.

Problems which can be solved by a polynomial-time algorithm are called tractable problem.

Certain computations, although theoretically possible, have such high resource requirements that in practice they must be rejected as unrealistic on existing computers, as well as on supercomputers yet to be designed. Such problems are sometimes called ***intractable*** to indicate that, while in principle computable, there is no realistic hope of a practical algorithm. To understand this better, computer scientists have attempted to put the idea of intractability on a formal basis. One attempt to define the term intractable is made in what is generally called the ***cook-karp thesis.*** In the cook-karp thesis, a problem that is in $P$ is called tractable, and one that is not is said to be intractable.

***Q.44. Write a brief note on untractable problems.*** ***(R.G.P.V., Dec. 2009, 2012)***

***Or***

***Write a short note on intractable problem.*** ***(R.G.P.V., Dec. 2007)***

***Ans.*** Refer the ans. of Q.43.

***Q.45. Define the term class P.***

***Or***

***Explain P-class problems with suitable example.*** ***(R.G.P.V., June 2015)***

*Ans.* $P$ is the class of problems which can be solved by a single tape deterministic turing machine in polynomial time.

So,

$$P = \bigcup_{k \geq 1} \text{DTIME}(n^k)$$

Example of P-class problem is – The PATH problem (Is there a path from s to t ?)

***Q.46. Define the term class NP.***

***Or***

***Explain NP class problem with suitable example. (R.G.P.V., Dec. 2014)***

*Ans.* NP is the class of problem which can be solved by a non-deterministic turing machine or it is a class of those languages which can be accepted by a non-deterministic turing machine in polynomial time.

So,

$$\text{NP} = \bigcup_{k>1} \text{NTIME}(n^k)$$

The types of NP-class problems are given below –

(i) NP-complete (ii) NP-hard.

Some examples of NP class problems are as follows –

(i) Travelling salesman problem
(ii) Graph colouring problem
(iii) Hamiltonian problem
(iv) Vertex cover problem
(v) Subset sum problem.

***Q.47. Explain P and NP type of problem. Write any three examples of P or NP type problem.*** ***(R.G.P.V., Dec. 2017)***

*Ans.* Refer to Q.45 and Q.46.

***Q.48. Define the relation between P and NP.***

***Or***

***How P class problems different from NP class problem ?***

***(R.G.P.V., Dec. 2016)***

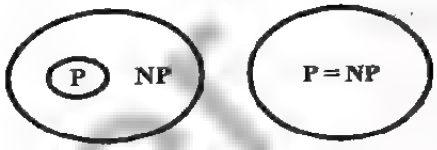
*Ans.* Every P-class problem is also the NP class problem but the reverse is fundamental unsolvable problem.

So, $P \subseteq NP$

P is the class of language for which membership can be decided quickly and NP is the class of language for which membership can be verified quickly



whether $NP \subseteq P$ or $P = NP$ is the fundamental unsolvable problem in theory of computation. It is not known whether there exist some language in NP, which does not have a deterministic turing machine accepting it. Fig. 5.32 shows two possibilities of relationship between P and NP, in which only one is correct.




*(a)* *(b)*

***Fig. 5.32 Either (a) or (b) is Correct***

***Q.49. Explain P and NP problems.*** ***(R.G.P.V., June 2016)***

***Or***

***Define P and NP problems.*** ***(R.G.P.V., Dec. 2015)***

*Ans.* Refer to Q.45, Q.46 and Q.48.

***Q.50. The set of all languages whose complements are in NP is called CO-NP. Prove that NP = CO-NP if and only if there is some NP-complete problem whose complement is in NP.*** ***(R.G.P.V., Dec. 2010)***

*Ans.* The "only if " part is obvious. For the "if " part let $S$ be an *NP*-complete problem, and suppose $\overline{S}$ were in *NP*. Since each $L$ in *NP* is log-space reducible to $S$, each $\overline{L}$ is log-space reducible to $\overline{S}$. Thus, $\overline{L}$ is in *NP*.

***Q.51. Write short note on NP-complete vs NP-hard problem.*** ***(R.G.P.V., Dec. 2013)***

***Or***

***Difference between NP-complete vs NP-hard problem.*** ***(R.G.P.V., June 2016)***

*Ans.* The study of the relation between the complexity classes *P* and *NP* has generated particular interest among computer scientists. At the root, this is the question whether or not –

$$P = NP$$

This is one of the fundamental unsolved problem in the theory of computation. To explore it, computer scientists have introduced a variety of related concepts. One of them is the idea of an *NP*-complete problem. Loosely speaking, an *NP*-complete problem is one that is as hard as any *NP* problem and in some sense is equivalent to all of them.

A language $L_1$ is said to be ***polynomial-time reducible*** to some language $L_2$ if there exists a deterministic Turing machine by which any $u_1$ in the

alphabet of $L_1$ can be transformed in polynomial time to a $u_2$ in the alphabet of $L_2$ in such a way that $u_1 \in L_1$ iff $u_2 \in L_2$.

We can see that if $L_1$ polynomial-time reducible to $L_2$, and if $L_2 \in P$, then $L_1 \in P$. Similarly, if $L_2 \in NP$, then $L_1 \in NP$.

A language $L \subseteq \{0, 1\}^*$ is ***NP-complete*** if

(i) $L \in NP$, and

(ii) $L' \leq_p L$ for every $L' \in NP$.

If a language satisfies the property (ii), but not necessarily property (i), then $L$ is ***NP-hard.***

It follows easily from these definitions that if some $L_1$ is *NP*-complete and polynomial-time reducible to $L_2$, then $L_2$ is also *NP*-complete. The implication of this definition is that if we can find a deterministic polynomial-time algorithm for any *NP*-complete language, then every language in *NP* is also in *P*, i.e.,

$$P = NP.$$

***Q.52. Define the following –***

***(i) Polynomial time reduction and NP-completeness***

***(ii) Cook's theorem.***

***(R.G.P.V., Dec. 2010)***

***Ans. (i) Polynomial Time Reduction and NP-completeness*** – Refer to Q.51.

***(ii) Cook's Theorem*** – The statement that the satisfiability problem is *NP*-complete is known as Cook's theorem.

***Q.53. Explain Cook's theorem.*** ***(R.G.P.V., Dec. 2011)***

***Ans.*** Refer to Q.52 (ii).

***Q.54. Explain P, NP, NP complete problems with example.***

***(R.G.P.V., Dec. 2012)***

***Ans.*** Refer to Q.45, Q.46 and Q.51.

***Q.55. What do you understand by P, NP, NP complete and NP hard problems.*** ***(R.G.P.V., June 2010)***

***Or***

***Define the following –***

***(i) Polynomial problems***

***(ii) Non-deterministic polynomial (NP) problems***

***(iii) NP-complete problems***

***(iv) NP-hard problems.***

***(R.G.P.V., Dec. 2010)***

***Ans.*** Refer to Q.45, Q.46 and Q.51.



***Q.56. Discuss the relationship between class P, NP, NP complete and NP hard problems with example of each class.***

***Or***

***Explain relation between P, NP, NP hard and NP complete problems with diagram.*** ***(R.G.P.V., Dec. 2011)***

***Or***

***Draw and explain commonly believed relationship between class P, NP, NP-complete and NP-hard.*** ***(R.G.P.V., Dec. 2015)***

***Ans.*** Refer to Q.48 for *P* and *NP*.

A problem $L$ is *NP*-hard if and only if satisfiability reduce to $L$ (satisfiability $\propto L$). A problem is *NP*-complete if and only if $L$ is *NP*-hard and $L \in NP$. All *NP*-complete problems are *NP*-hard but all *NP*-hard problems cannot be *NP*-complete. If an *NP*-hard problem can be solved in polynomial time then all *NP*-complete problems can also be solved in polynomial time.

Only a decision problem can be *NP*-complete. However, an optimization problem may be *NP*-hard. Furthermore if $L_1$ is a decision problem and $L_2$ an optimization problem, then it is possible that $L_1 \propto L_2$. One can trivially show that the knapsack decision problem reduces to the knapsack optimization problem. For the clique problem one can easily show that the clique decision problem reduces to the clique optimization problem. In fact, one can also show that these optimization problems reduce to their corresponding decision problems. Fig. 5.33 shows the relationship among these classes.

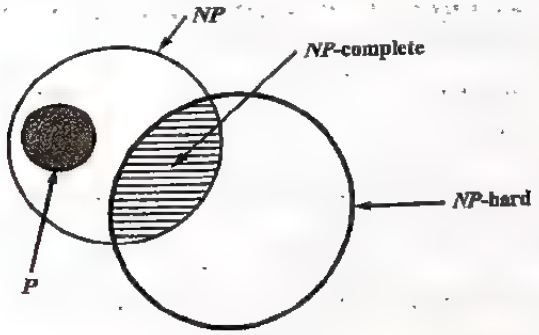



***Fig. 5.33***

***Q.57. What is NP hard problem ?*** ***(R.G.P.V., June 2011, Dec. 2016)***

***Or***

***What are NP hard problems ? Write a brief note on it.*** ***(R.G.P.V., Dec. 2009)***

***Or***

***Explain what is NP-hard problem.*** ***(R.G.P.V., June 2015)***

***Ans.*** Refer to Q.51 and Q.56.

***Q.58. Write short notes (any three) –***

***(i) Recursive and recursively enumerable language***

***(ii) Halting problem***

***(iii)NP-complete vs NP-hard***

***(iv) CNF***

***(v) NPDA.***

***(R.G.P.V., Nov. 2018)***

***Ans. (i) Recursive and Recursively Enumerable Language*** – Refer to Q.30.

***(ii) Halting Problem*** – Refer to Q.41.

***(iii)NP-Complete vs NP-hard*** – Refer to Q.51.

***(iv) CNF*** – Refer to Q.12, Unit-III.

***(v) NPDA*** – Refer to Q.12, Unit-IV.

***Q.59. Explain satisfiability problem.***

***Or***

***Write short note on satisfiability problem.*** ***(R.G.P.V., June 2015)***

***Ans.*** An expression is satisfiable if there is some assignment of 0's and 1's to the variables that gives the expression the value 1. The satisfiability problem is to determine, given a Boolean expression, whether it is satisfiable.

We may represent the satisfiability problem as a language $L_{sat}$ as follows –

Let the variables of some expression be $x_1, x_2 \ldots\ldots x_m$ for some $m$. Code $x_i$ as the symbol $x$ followed by $i$ written in binary. The alphabet of $L_{sat}$ is thus.

$$\{\wedge, \vee, \neg, (,), x, 0, 1\}.$$

The length of the coded version of an expression of $n$ symbols is easily seen to be no more than $[n \log_2 n]$, since each symbol other than a variable is coded by one symbol, there are no more than $[n/2]$ different variables in an expression of length $n$, and the code for a variable requires no more than $1 + [\log_2 n]$ symbols. We shall henceforth treat the word in $L_{sat}$ representing an expression of length $n$ as if the word itself ware of length $n$. Our results will not depend on whether we use $n$ or $n \log n$ for the length of the word, since $\log (n \log n) \leq 2 \log n$, and we shall deal with log space reductions.

***Q.60. Show that the satisfiability problem is NP-complete.***

***Ans.*** To determine if an expression of length $n$ is satisfiable, non-deterministically guess values for all the variables and then evaluate the expression. Thus $L_{sat}$ is in $NP$.



To show that every language in $NP$ is reducible to $L_{sat}$, for each NTM $M$ that is time bounded by a polynomial $p(n)$, we give a log-space algorithm that takes as input a string $x$ and produces a Boolean formula $E_x$ that is satisfiable if and only if $M$ accepts $x$. We now describe $E_x$.

Let $\# \beta_0 \# \beta_1 \# \ldots\ldots \# \beta_{p(n)}$ be a computation of $M$, where each $\beta_i$ is an ID consisting of exactly $p(n)$ symbols. If acceptance occurs before the $p(n)$th move, we allow the accepting ID to repeat, so each computation has exactly $p(n) + 1$ ID's. In each ID we group the state with the symbol scanned to form a single composite symbol. Numbers are assigned to moves by arbitrarily ordering the finite set of choices that $M$ may make given a state and tape symbol.

For each symbol that can appear in a computation and for each $i$; $0 \leq i < (p(n) + 1)^2$, we create a Boolean variable $C_{ix}$ to indicate whether the $i$th symbol in the computations is $X$. The expression $E_x$ that we shall construct will be true for a given assignment to the $C_{ix}$'s if and only if the $C_{ix}$'s that are true correspond to a valid computations.

The expression $E_x$ states the following –

(i) The $C_{ix}$'s that are true correspond to a string of symbols in that exactly one $C_{ix}$ is the true for each $i$.

(ii) The ID $\beta_0$ is an initial ID of $M$ with input $x$

(iii) The last ID contains a final state

(iv) Each ID follows from the previous one by the move of $M$ that is indicated.

The first formula, stating that for each $i$ between 0 and $(p(n) + 1)^2 - 1$ exactly one $C_{ix}$ is true is

$$\bigwedge_i \left[ \bigvee_x C_{ix} \wedge \neg \left( \bigvee_{x \neq Y} (C_{ix} \wedge C_{iY}) \right) \right]$$

For a given value of $i$ the term $\vee x C_{ix}$ forces at least one $C_{ix}$ to be true and $\neg \vee_{x \neq Y} (C_{ix} \wedge C_{iY})$ forces at most one to be true.

Let $x = a_1, a_2, \ldots\ldots a_n$. The second formula expressing the fact that $\beta_0$ is an initial $I_D$ is in turn the AND of the following terms.

(i) $C_{0\#} \wedge C_{p(n)+1\#}$. The symbols in positions 0 and $p(n) + 1$ are #

(ii) $C_{1Y1} \vee C_{1Y2} \vee \ldots\ldots \vee C_{1YK}$, where $Y_1, Y_2 \ldots\ldots Y_K$ are all the composite symbols that represent tape symbol $a_1$ the start state $q_0$, and the number of a legal move of $M$ in state $q_0$ reading symbol $a_1$. This clause states that the first symbol of $\beta_0$ is correct

(iii) $\bigwedge_{2 \leq i \leq n} C_i a_i$. The 2nd through $n$th symbols of $\beta_0$ are correct

(iv) $\bigwedge_{n \leq i \leq p(n)} C_{iB}$. The remaining symbols of $\beta_0$ are blank.

The 3rd formula says that the last ID has an accepting state. It can be written as

$$\bigvee_{p(n)(p(n)+1)<i<(p(n)+1)^2}\left(\bigvee_{X \text{ in } F} C_{ix}\right)$$

where $F$ is the set of composite symbols that include a final state.

A predicate $f(W, X, Y, Z)$ that is true if and only if symbol $Z$ could appear in position $j$ of some ID given that $W$, $X$ and $Y$ are the symbols in positions $j-1$, $j$, and $j+1$ of the previous ID [$W$ is # if $j = 1$ and $Y$ is # if $j = p(n)$]. It is convenient also to declare $f(W, \#, X, \#)$ to be true, so we can treat the markers between ID's as we treat the symbol within ID's. We can now express the fourth formula as –

$$\bigwedge_{p(n)\le j<(p(n)+1)^2}\left(\bigwedge_{\substack{W,X,Y,Z \text{ such} \\ \text{that } f(W,X,Y,Z)}}\left(C_{j-p(n)-2W,}\wedge C_{j-p(n)-1,X}\wedge C_{j-p(n),Y}\wedge C_{jZ}\right)\right)$$

Given an assignment of truth values making $E_x$ true, the four formulas above guarantee that there is an accepting computation of $M$ on $x$.

The formulas composing $E_x$ are of length $O(p^2(n))$ and are sufficiently simple that a log-space TM can generate them given $x$ on its input, the TM only needs sufficient storage to count upto $(p(n)+1)^2$.

Since the logarithm of a polynomial in $n$ is some constant times log $n$, this can be done with $O$ (log $n$) storage. We have thus shown that every language in $NP$ is log-space reducible to $L_{sat}$, proving that $L_{sat}$ is $NP$-complete.

***Q.61. Discuss satisfiability problem. Prove that satisfiability is NP complete.*** ***(R.G.P.V., Dec. 2009)***

***Ans.*** Refer to Q.59 and Q.60.

***Q.62. Show that the $L_{csat}$, the satisfiability problem for CNF expressions is NP-complete.***

***Ans.*** Clearly $L_{csat}$ is in $NP$, since $L_{sat}$ is. We reduce $L_{sat}$ to $L_{csat}$ as follows. Let $E$ be an arbitrary Boolean expression of length $n$. Certainly, the number of variable occurrences in $E$ does not exceed n, nor does the number of $\wedge$ and $\vee$ operators. Using the identities –

$$\left.\begin{aligned}\neg(E_1 \wedge E_2) &= \neg(E_1) \vee \neg(E_2) \\ \neg(E_1 \vee E_2) &= \neg(E_1) \wedge \neg(E_2) \\ \neg\neg E_1 &= E_1\end{aligned}\right] \quad \ldots(i)$$



We can transform $E$ to an equivalent expression $E'$, in which the $\neg$ operators are applied only to variables, never to more complex expressions. The validity of equation (i) may be checked by considering the four assignments of values 0 and 1 to $E_1$ and $E_2$. Incidently, the first two of these equations are known as DeMorgan's laws.

We create $E''$, an expression in CNF that is satisfiable. If and only if $E'$ is satisfiable. Let $V_1$ and $V_2$ be sets of variables with $V_1 \subseteq V_2$.

We say an assignment of values to $V_2$ is an extension of an assignment of values to $V_1$. If the assignments agree on the variable of $V_1$. We shall prove by induction on $r$, the number of $\wedge$'s and $\vee$'s in an expression $E'$, all of whose negations are applied to variables, that if $|E'| = n$, then there is a list of at most $n$ clauses, $F_1, F_2 \ldots\ldots F_k$ over a set of variables that includes the variables of $E'$ and at most $n$ other variables, such that $E'$ is given value 1 by an assignment to variables if and only if there is an extension of that assignment that satisfies $F_1 \wedge F_2 \wedge \ldots\ldots \wedge F_k$.

Basis $r = 0$, then $E'$ is a literal, and we may take that literal in a clause by itself to satisfy the condition.

If $E' = E_1 \wedge E_2$, Let $F_1, F_2 \ldots\ldots F_k$ and $G_1, G_2 \ldots\ldots G_l$ be the clauses for $E_1$ and $E_2$ that exist by the inductive hypothesis. Assume without loss of generality that no variable that is not present in $E'$ appears both among the $F$'s and among the $G$'s. Then $F_1, F_2 \ldots\ldots F_k, G_1, G_2 \ldots\ldots G_l$ satisfies the condition for $E'$. If $E' = E_1 \vee E_2$, let the $F$'s and $G$'s be as above, and let $y$ be a new variable. Then $Y \vee F_1, Y \vee F_2 \ldots\ldots Y \vee F_k, \bar{y} \vee G_1, \bar{y} \vee G_2 \ldots\ldots \bar{y} \vee G_2$ satisfies the conditions.

Suppose all the clauses for $E'$ are satisfied by some assignment, if that assignment sets $y = 1$, then all of $G_1, G_2 \ldots\ldots G$ must be satisfied, so $E_2$ is satisfied. A similar argument applies if $y = 0$, the desired expression $E''$ is all the clauses for $E'$ connected by $\wedge$'s .

Let $y_i$ be the variables introduced by the $i$th $V$. The final expression is the logical AND of clauses, where each clause contains a literal of the original expression. In addition, if the literal is in the left subtree of the $i$th $\vee$, then the clause also contains $y_i$, if the literal is in the right subtree of the $i$th $\vee$, then the clause contains $\bar{y}_i$. The input is scanned from left to right. To determine which $y_i$'s and $\bar{y}_i$'s to include in the clause, we use a counter of length log $n$ to remember our place on the input. We can scan the entire input, and for $\vee$ symbol, say the $i$th from the left, we determine its left and right operands, using another counter of length log $n$ to count parentheses. If the current literal is in the left operand, generate $y_i$, if it is in the right operand, generate $\bar{y}_i$ and if in neither operand, generate neither $y_i$ nor $\bar{y}_i$.

We have thus reduced each Boolean expression $E$ to a CNF expression $E'$ that is in $L_{csat}$ if and only if $E$ is in $L_{sat}$. Since the reduction is accomplished in log-space, the *NP*-completeness of $L_{sat}$ implies the *NP*-completeness of $L_{csat}$.

***Q.63. Prove that the $L_{3sat}$, the satisfiability problem for 3-CNF expression, is NP-complete.***

***Ans.*** Clearly, $L_{3sat}$ is in *NP*, since $L_{sat}$ is. Let $E = F_1 \wedge F_2 \ldots\ldots \wedge \ldots\ldots \wedge F_k$ be a CNF expression. Suppose some clause $F_i$ has more than three literals say,

$F_i = \alpha_1 \vee \alpha_2 \vee \ldots\ldots \vee \alpha_l$ ; $l > 3$ Introduce new variables $y_1, y_2 \ldots\ldots y_{l-3}$ and replace $F_i$ by $(\alpha_1 \vee \alpha_2 \vee y_1) \wedge (\alpha_3 \vee \bar{y}_1 \vee y_2) \wedge (\alpha_4 \vee \bar{y}_2 \vee y_3) \wedge \ldots\ldots \wedge (\alpha_{l-2} \vee \bar{y}_{l-4} \vee y_{l-3}) \wedge (\alpha_{l-1} \vee \alpha_l \vee \bar{y}_{l-3})$ ....(i)

Then $F_i$ is satisfied by an assignment if and only if an extension of the assignment satisfies (i). An assignment satisfying $F_i$ must have $\alpha_j = 1$ for some $j$. Thus assume that the assignment gives literals $\alpha_1, \alpha_2 \ldots\ldots \alpha_{j-1}$ the value 0 and $\alpha_j$ the value 1. Then $y_m = 1$ for $m \leq j-2$ and $y_m = 0$ for $m \geq j-1$ is an extension of the assignment satisfying (i).

Conversely, we must show that any assignment satisfying (i) must have $\alpha_j = 1$ for some $j$ and thus satisfies $F_i$. Assume to the contrary that the assignment gives all the $\alpha_m^{as}$ the value 0. Then since the first clause has value 1, it follows that $y_1 = 1$. Since the second clause has value 1, $y_2$ must be 1 and by induction $y_m = 1$ for all $m$. But then the last clause would have the value 0, contradicting the assumption that (i) is satisfied. Thus any assignment that satisfies (i) also satisfies $F_i$.

The only other alterations necessary are when $F_i$ consists of one or two literals. In the latter case replace $\alpha_1 \vee \alpha_2$ by $(\alpha_1 \vee \alpha_2 \vee y) \wedge (\alpha_1 \vee \alpha_2 \vee \bar{y})$, where $y$ is a new variable, and in the former case an introduction of two new variables suffices. Thus $E$ can be converted to a 3-CNF expression that is satisfiable if and only if $E$ is satisfiable. The transformation is easily accomplished in log-space. We have thus a log-space reduction of $L_{csat}$ to $L_{3sat}$ and conclude that $L_{3sat}$ is *NP*-complete.

***Q.64. Explain vertex cover problem.*** ***(R.G.P.V., Dec. 2012)***

***Or***

***Write short note on vertex cover problem.*** ***(R.G.P.V., Dec. 2013)***

***Or***

***Explain vertex cover problem with suitable example.*** ***(R.G.P.V., June 2015)***

***Or***

***Define and discuss vertex cover problem.*** ***(R.G.P.V., Dec. 2015)***



***Ans.*** Let $G = (V, E)$ be an (undirected) graph with set of vertices $V$ and edges $E$. A subset $A \subseteq V$ is said to be vertex cover of $G$ if for every edge $(v, w)$ in $E$, at least one of $v$ or $w$ is in $A$.

The vertex cover problem is – given a graph $G$ and integer $k$, does $G$ have a vertex cover of size $k$ or less ?

To represent this problem as a language $L_{vc}$, consisting of strings of the form– $k$ in binary, followed by a marker, followed by the list of vertices, where $v_i$ is represented by $v$ followed by $i$ in binary, and a list of edges, where $(v_i, v_j)$ is represented by the codes for $v_i$ and $v_j$ surrounded by parentheses. $L_{vc}$ consists of all such strings representing $k$ and $G$, such that $G$ has a vertex cover of size $k$ or less.

***Q.65. Prove that the $L_{vc}$, the vertex cover problem, is NP-complete.***

***Or***

***Prove that vertex cover is NP-complete problem.*** ***(R.G.P.V., June 2011, Dec. 2011)***

***Or***

***Prove that vertex cover problem is NP-complete problem.*** ***(R.G.P.V., Dec. 2014)***

***Ans.*** Let $A$ subset of $k$ vertices and check that it covers all edges. This may be done in time proportional to the square of the length of the problem representation. $L_{vc}$ is shown to be *NP*-complete by reducing 3-CNF satisfiability to $L_{vc}$.

Let $F = F_1 \wedge F_2 \wedge \ldots\ldots\ldots\ldots \wedge F_q$ be an expression in 3-CNF, where each $F_i$ is a clause of the form $(\alpha_{i1} \vee \alpha_{i2} \vee \alpha_{i3})$ each $\alpha_{ij}$ being a literal. We construct an undirected graph $G = (V, E)$ whose vertices are pairs if integers $(i, j)$, $1 \leq i \leq 2$, $1 \leq j \leq 3$. The vertex $(i, j)$ represents the $j$th literal of the $i$th clause. The edges of the graph are

(i) $[(i, j), (i, k)]$ provided $j \neq k$ and

(ii) $[(i, j), (k, l)]$ if $\alpha_{ij} = \neg\, \alpha_{kl}$.

Each pair of vertices corresponding to the same clauses are connected by an edge in (i). Each pair of vertices corresponding to a literal and its complement are connected by an edge in (ii).

$G$ has been constructed so that it has a vertex cover of size $2q$ if and only if $F$ is satisfiable. Assume $F$ is satisfiable and fix an assignment satisfying $F$. Each clause must have a literal whose value is 1. Select one such literal for each clause. Delete the $q$ vertices corresponding to these literal from $V$. The remaining vertices form a vertex cover of size $2q$. Clearly for each $i$, only one vertex of the form $(i, j)$ is missing from the cover, and hence each edge in (i)

is incident upon at least one vertex in the cover. Since edges in (ii) are incident upon two vertices corresponding to some literal and its complement, and since we would not have deleted both a literal and its complement, one or the other of these vertices is in the cover. Thus we indeed have a cover of size $2q$.

Conversely, assume we have a vertex cover of size $2q$, for each $i$ the cover must contain all but one vertex of the form $(i, j)$ for if two such vertices were missing, an edge $[(i, j), (i, k)]$ would not be incident upon any vertex in the cover. There can be no conflict because two vertices not in the cover cannot correspond to a literal and its complement, else there would be an edge in group (ii) not incident upon any vertex of the cover. For this assignment $F$ has value 1. Thus $F$ is satisfiable. We can essentially use the variables names in the formula $F$ as the vertices of $G$, appending two bits for the $j$-component in vertex $(i, j)$. Edges of type (i) are generated directly from the clauses, while those of type (ii) require two counters to consider all pairs of literals. Thus we conclude that $L_{vc}$ is *NP*-complete.

***Q.66. Describe vertex cover problem.*** ***(R.G.P.V., June 2016)***

***Or***

***Explain vertex cover problem briefly.*** ***(R.G.P.V., Dec. 2016)***

***Ans.*** Refer to Q.64 and Q.65.

***Q.67. What is Hamilton circuit problem ? Prove that directed Hamilton circuit is NP-complete.*** ***(R.G.P.V., Dec. 2009)***

***Or***

***What is Hamiltonian path problem ? Discuss with example.*** ***(R.G.P.V., June 2011)***

***Or***

***Write short note on Hamiltonian path problem.*** ***(R.G.P.V., Dec. 2013, June 2015)***

***Or***

***Describe Hamiltonian path problem.*** ***(R.G.P.V., June 2016)***

***Or***

***What is Hamiltonian path problem ? Explain.*** ***(R.G.P.V., Dec. 2016)***

***Ans.*** The Hamiltonian circuit problem is – Given a graph $G$, does $G$ contain path that visits each vertex exactly once and returns to its starting point.

We first show that HAM-CYCLE belongs to *NP*. Given a graph $G = (V, E)$, our certificate is the sequence of $|V|$ vertices that make up the Hamiltonian cycle. The verification algorithm checks that this sequence contains each vertex in $V$ exactly once and that with the first vertex repeated at that end, if forms a cycle in $G$. This verification can be performed in polynomial time.



We now prove that HAM-CYCLE is *NP*-complete by showing that 3 CNF-SAT $\leq_p$ HAM-CYCLE. Given a 3-CNF boolean formula $\phi$ over variables $x_1, x_2, \ldots, x_n$ with clauses $C_1, C_2, \ldots, C_k$ each containing exactly 3 distinct literals, we construct a graph $G = (V, E)$ in polynomial time such that $G$ has a Hamiltonian cycle if and only if $\phi$ is satisfiable. Our construction is based on ***widgets***, which are pieces of graphs that enforce certain properties.

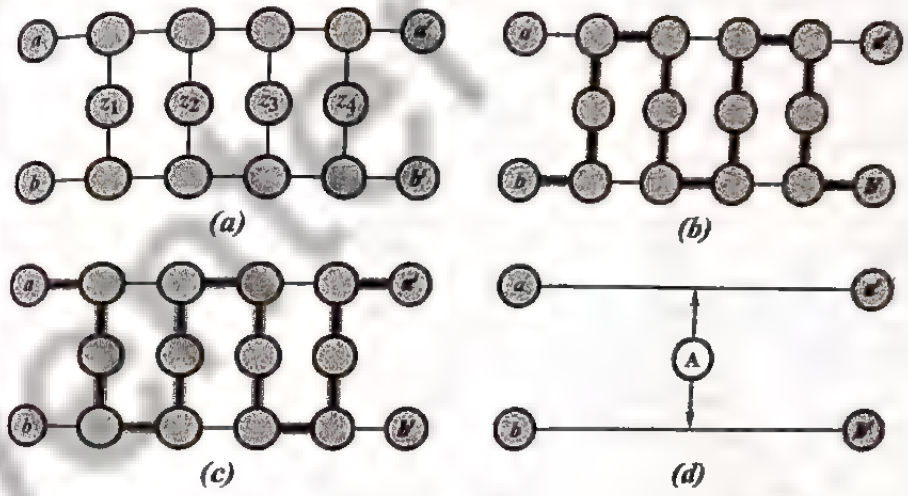



***Fig. 5.34***

Our first widget is the subgraph $A$ shown in fig. 5.34 (a). Suppose that $A$ is a subgraph of some graph $G$ and that the only connections between $A$ and the remainder of $G$ are through the vertices $a$, $a'$, $b$ and $b'$. Furthermore, suppose that graph $G$ has a Hamiltonian cycle. Since any Hamiltonian cycle of $G$ must pass through vertices $z_1$, $z_2$, $z_3$ and $z_4$ in one of the ways shown in fig. 5.34 (b) and (c), we may treat subgraph $A$ as if it were simply a pair of edges $(a, a')$ and $(b, b')$ with the restriction that any Hamiltonian cycle of $G$ must include exactly one of these edges. We shall represent widget $A$ as shown in fig. 5.34 (d).

The subgraph $B$ in fig. 5.35 is our second widget. Suppose that $B$ is a subgraph of some graph $G$ and that the only connections from $B$ to the remainder of $G$ are through vertices $b_1$, $b_2$, $b_3$ and $b_4$. A Hamiltonian cycle of graph $G$ cannot traverse all of the edges $(b_1, b_2)$, $(b_2, b_3)$ and $(b_3, b_4)$, since then all vertices in the widget other than $b_1$, $b_2$, $b_3$ and $b_4$ would be missed. However, a Hamiltonian cycle of $G$ may traverse any proper subset of these edges. Fig. 5.35 (a) to (e) show five such subsets; the remaining two subsets can be obtained by performing a top-to-bottom flip of parts (b) and (e). This widget is represented as in fig. 5.35 (f), the idea being that at least one of the paths pointed to by the arrows must be taken by a Hamiltonian cycle.

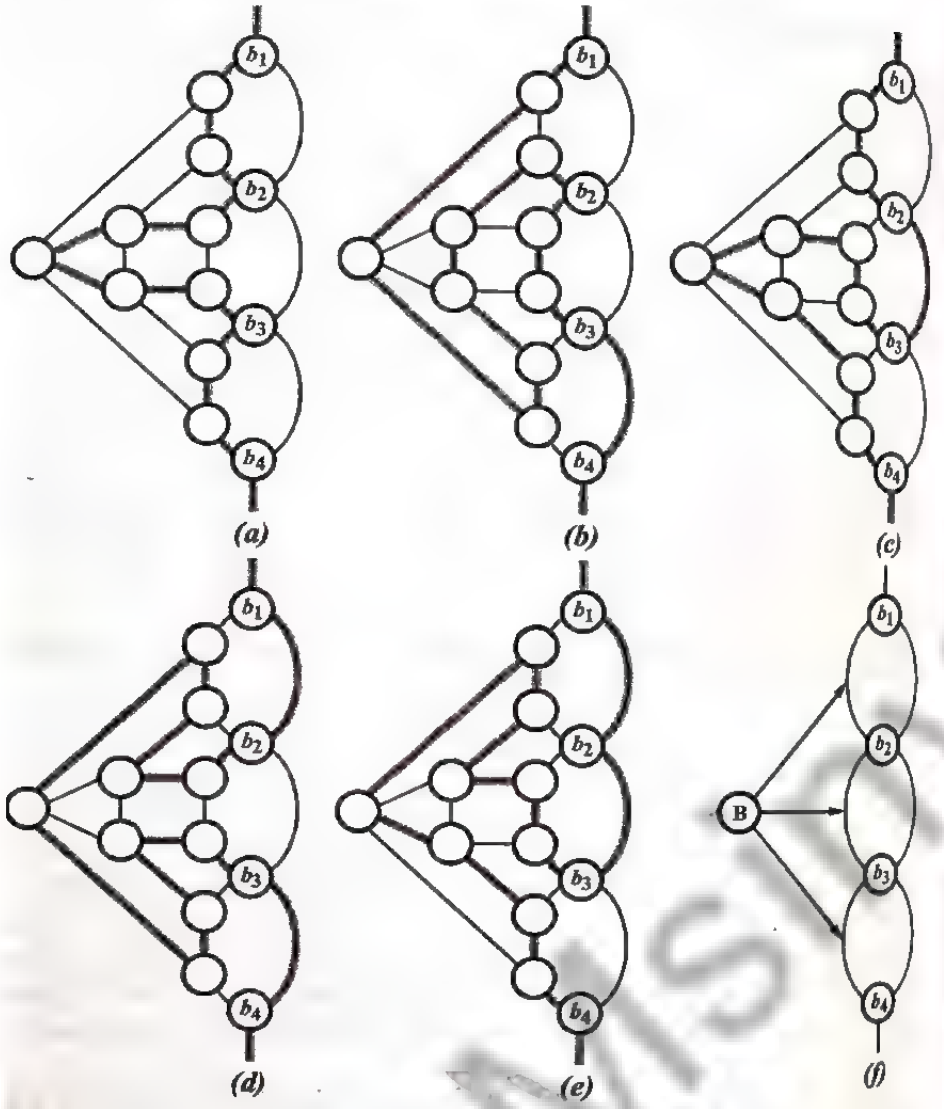



*Fig. 5.35*

The graph $G$ that we shall construct consists mostly of copies of these two widgets. The construction is shown in fig. 5.36. For each of the $k$ clauses in $\phi$, we include a copy of widget $B$, and we join these widgets together in series as follows. Letting $b_{ij}$ be the copy of vertex $b_j$ in the $i^{th}$ copy of widget $B$, we connect $b_{i,4}$ to $b_{i+1,1}$ for $i = 1, 2, ....., k-1$.

Then for each variable $x_m$ in $\phi$, we include two vertices $x'_m$ and $x''_m$. These two vertices are connected by means of two copies of the edge $(x'_m, x''_m)$, which are denoted by $e_m$ and $\bar{e}_m$. The idea is that if the Hamiltonian



cycle takes edge $e_m$, it corresponds to assigning variable $x_m$ the value 1. If the Hamiltonian cycle takes edge $\bar{e}_m$ the variable is assigned the value 0. Each pair of these edges forms a two-edge loop; these small loops are connected in series by adding edges $(x'_m, x''_{m+1})$ for $m = 1, 2, ...., n-1$. The left (clause) side of the graph is connected to the right (variable) side by means of two edges $(b_{1,1}, x'_1)$ and $(b_{k,4}, x''_n)$ which are the topmost and bottommost edges in fig. 5.36.

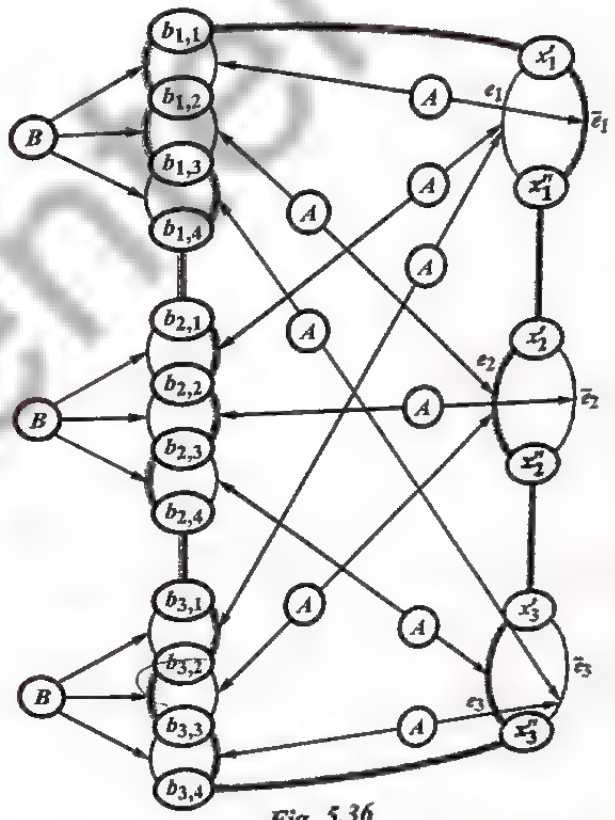


*Fig. 5.36*

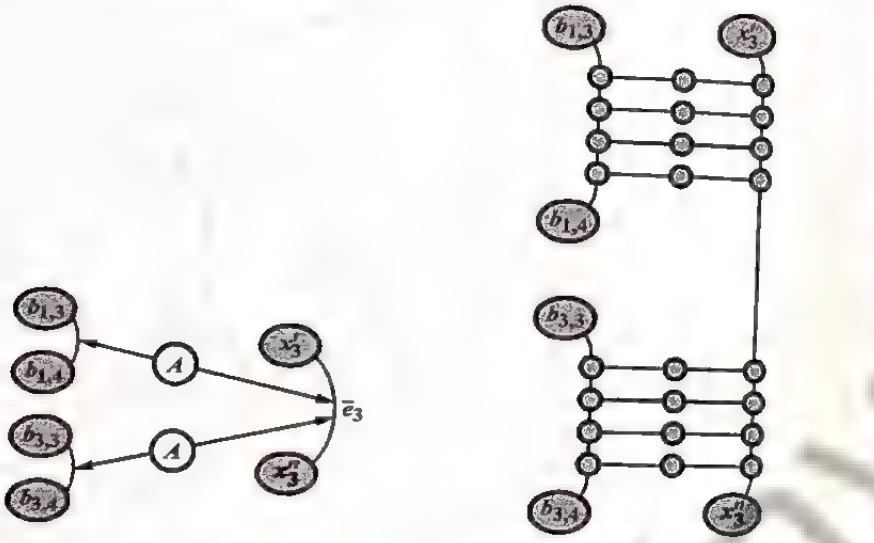
We are not yet finished with the construction of graph $G$, since we have yet to relate the variables to the clauses. If the $j^{th}$ literal of clause $c_i$ is $x_m$, then we use an $A$ widget to connect edge $(b_{ij}, b_{i,j+1})$ with edge $e_m$. If the $j^{th}$ literal of clause $c_i$ is $\neg x_m$, then we instead put an $A$ widget between edge $(b_{ij}, b_{i,j+1})$ and edge $\bar{e}_m$. In fig. 5.36, for example, because clause $C_2$ is $(x_1 \vee \neg x_2 \vee x_3)$, we place three $A$ widgets as follows –

(i) between $(b_{2,1}, b_{2,2})$ and $e_1$

(ii) between $(b_{2,2}, b_{2,3})$ and $\bar{e}_2$

(iii) between $(b_{2,3}, b_{2,4})$ and $e_3$.

Note that connecting two edges by means of $A$ widgets actually entails replacing each edge by the five edges in the top or bottom of fig. 5.34 (a) and, of course, adding the connections that pass through the $z$ vertices as well. A given literal $l_m$ may appear in several clauses (for example, $\neg x_3$ in fig. 5.36), and thus an edge $e_m$ or $\bar{e}_m$ may be influenced by several $A$ widgets (for example, edge $\bar{e}_3$). In this case, we connect the $A$ widgets in series, as shown in fig. 5.37, effectively replacing edge $e_m$ or $\bar{e}_m$ by a series of edges.




*(a) A Portion of Fig. 5.36* *(b) The actual Subgraph Constructed*

***Fig. 5.37***

We claim that formula $\phi$ is satisfiable if and only if graph $G$ contains a Hamiltonian cycle. We first suppose that $G$ has a Hamiltonian cycle $h$ and show that $\phi$ is satisfiable. Cycle $h$ must take a particular form –

(i) First, it traverses edge $(b_{1,1}, x'_1)$ to go from the top left to the top right.

(ii) It then follows all of the $x'_m$ and $x''_m$ vertices from top to bottom, choosing either edge $e_m$ or edge $\bar{e}_m$, but not both.

(iii) It next traverses edge $(b_{k,4}, x''_n)$ to get back to the left side.

(iv) Finally, it traverses the $B$ widgets from bottom to top on the left.

Given the Hamiltonian cycle $h$, we define a truth assignment for $\phi$ as follows. If edge $e_m$ belongs to $h$, then we set $x_m = 1$. Otherwise, edge $\bar{e}_m$ belongs to $h$, and we set $x_m = 0$.

We claim that this assignment satisfies $\phi$. Consider a clause $C_i$ and the corresponding $B$ widget in $G$. Each edge $(b_{i,j}, b_{i,j+1})$ is connected by an $A$



widget to either edge $e_m$ or edge $\bar{e}_m$, depending on whether $x_m$ or $\neg x_m$ is the $j^{th}$ literal in the clause. The edge $(b_{i,j}, b_{i,j+1})$ is traversed by $h$ if and only if the corresponding literal is 0. Since each of the three edges $(b_{i,1}, b_{i,2})$, $(b_{i,2}, b_{i,3})$, $(b_{i,3}, b_{i,4})$ in clause $C_i$ is also in a $B$ widget, all three cannot be traversed by the Hamiltonian cycle $h$. One of the three edges, therefore, must have a corresponding literal whose assigned value is 1, and clause $C_i$ is satisfied. This properly holds for each clause $C_i$, $i = 1, 2, ...., k$, and thus formula $\phi$ is satisfied.

Conversely, let us suppose that formula $\phi$ is satisfied by some truth assignment. By following the rules from above, we can construct a Hamiltonian cycle for graph $G$ – traverse edge $e_m$ if $x_m = 1$, traverse edge $\bar{e}_m$ if $x_m = 0$, and traverse edge $(b_{i,j}, b_{i,j+1})$ if and only if the $j^{th}$ literal of clause $C_i$ is 0 under the assignment. These rules can indeed be followed, since we assume that $s$ is a satisfying assignment for formula $\phi$.

Finally, we note that graph $G$ can be constructed in polynomial time. It contains one $B$ widget for each of the $k$ clauses in $\phi$. There is one $A$ widget for each instance of each literal in $\phi$, and so there are $3k$ $A$ widgets. Since the $A$ and $B$ widgets are of fixed size, the graph $G$ has $O(K)$ vertices and edges and is easily constructed in polynomial time. Thus, we have provided a polynomial-time reduction from 3-CNF-SAT to HAM-CYCLE.

***Q.68. What is travelling salesman problem ?***

***Ans.*** The travelling salesman problem is – Given $n$ cities, the distance between them and a number $D$, does there exists a tour programme for a salesman to visit all the cities so that the total distance travelled is almost $D$ ?

***Q.69. Show that the travelling salesman problem is NP-complete.***
***(R.G.P.V., Dec. 2010, 2013)***

***Ans.*** We first show that TSP (travelling-salesman problem) belongs to $NP$. Given an instance of the problem, we use as a certificate the sequence of $n$ vertices in the tour. The verification algorithm checks that this sequence contains each vertex exactly once, sums up the edge costs, and checks whether the sum is at most $k$. This process can certainly be done in polynomial time.

To prove that TSP is $NP$-hard, we show that HAM-CYCLE $\leq_p$ TSP. Let $G = (V, E)$ be an instance of HAM-CYCLE. We construct an instance of TSP as follows. We form the complete graph $G' = (V, E')$, where $E' = \{(i, j): i, j \in V\}$, and we define the cost function $c$ by

$$c(i, j) = \begin{cases} 0 & \text{if } (i, j) \in E, \\ 1 & \text{if } (i, j) \notin E. \end{cases}$$

The instance of TSP is then $(G', c, 0)$, which is easily formed in polynomial time.

We now show that graph $G$ has a Hamiltonian cycle if and only if graph $G'$ has a tour of cost at most 0. Suppose that graph $G$ has a Hamiltonian cycle $h$. Each edge in $h$ belongs to $E$ and thus has cost 0 in $G'$. Thus, $h$ is a tour in $G'$ with cost 0. Conversely, suppose that graph $G'$ has a tour $h'$ of cost at most 0. Since the costs of the edges in $E'$ are 0 and 1, the cost of tour $h'$ is exactly 0. Therefore, $h'$ contains only edges in $E$. We conclude that $h$ is a Hamiltonian cycle in graph $G$.

***Q.70. What is NP complete problems ? Show that travelling salesman problem is NP complete.*** ***(R.G.P.V., Dec. 2012)***

***Ans. NP* Complete Problems** – Refer to Q.51.

**Travelling Salesman Problem** – Refer to Q.69.

***Q.71. Discuss travelling salesman problem.*** ***(R.G.P.V., Dec. 2011)***

***Or***

***Explain travelling salesman problem.*** ***(R.G.P.V., June 2010)***

***Or***

***Discuss and explain travelling salesman problem.*** ***(R.G.P.V., Dec. 2015)***

***Ans.*** Refer to Q.68 and Q.69.

***Q.72. Write short notes on –***

***(i) NP-hard problem***

***(ii) Travelling salesman problem.***

***(R.G.P.V., Dec. 2014)***

***Ans. (i) NP-hard Problem*** – Refer to Q.51.

***(ii) Travelling Salesman Problem*** – Refer to Q.68 and Q.69.

***Q.73. What is the partition problem ?***

***Or***

***Discuss partition problem.*** ***(R.G.P.V., June 2011)***

***Ans.*** Given a list of integers $i_1, i_2 \ldots\ldots i_k$, does there exist a subset whose sum is exactly 1/2 $(i_1 + i_2 + \ldots. + i_k)$. Note that this problem appears to be in $P$ until we remember that the length of an instance is not $i_1 + i_2 + \ldots\ldots + i_k$ but the sum of the lengths of the $i_j$'s written in binary or some other fixed base.

●●●

RGPV

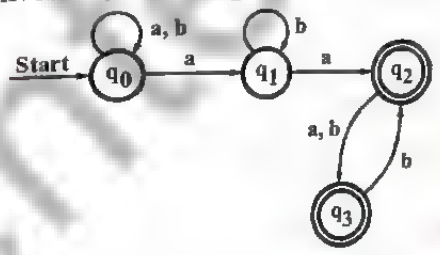
**B.E. (Fifth Semester) EXAMINATION, Dec. 2008**
**(Computer Science & Engg. Branch)**
**THEORY OF COMPUTATION**
**(CS–505)**

Note : Attempt any two questions from each Unit. All questions carry equal marks.

**Unit–I**

1. (a) Design a FA which accepts set of strings containing four 1's in every string over alphabet $\Sigma = \{0, 1\}$. (See Unit-I, Page 21, Prob.1)

(b) Convert the following NFA into DFA.




(See Unit-I, Page 35, Prob.23)

(c) Construct a Moore machine which calculates mod-4 for each binary string treated as binary integer. (See Unit-I, Page 38, Prob.28)

**Unit–II**

2. (a) Write regular expression for the following language –
The set of strings over alphabet {a, b, c} containing at least one a and at least one b. (See Unit-II, Page 63, Prob.3)

(b) Give english description of the language of the following regular expression –

$$(0^* 1^*)^* 000 (0 + 1)^*$$

(See Unit-II, Page 63, Prob.4)

(c) Convert the following regular expression to NFA* with ∈ transition –

$$01^*$$

(See Unit-II, Page 63, Prob.5)

**Unit–III**

3. (a) Prove that the following is not regular language –

$$\{0^n 1^n | n \geq 1\}$$

This language consisting of a string of 0's followed by an equal length string of 1's, is the language $L_{01}$. (See Unit-II, Page 81, Prob.27)

(b) Design context free grammars for the following languages –
The set $\{0^n 1^n | n \geq 1\}$ i.e., the set of all strings of one or more 0's followed by an equal number of 1's. (See Unit-III, Page 147, Prob.52)

(c) Consider the grammar –

$$S \rightarrow aS \mid aSbS \mid \in$$

This grammar is ambiguous. Show in particular that the string aab

Write short notes –

(i) Satisfiability problem (See Unit-V, Page 252, Q.59)

(ii) Hamiltonian path problem. (See Unit-V, Page 258, Q.67)

RGPV

**B.E. (Fifth Semester) EXAMINATION, Dec. 2015**
**THEORY OF COMPUTATION**
**(CS-505)**

**Note :** (i) Answer five questions. In each question part A, B, C is compulsory and D part has internal choice.

(ii) All parts of each questions are to be attempted at one place.

(iii) All questions carry equal marks, out of which part A and B (Max. 50 words) carry 2 marks, part C (Max. 100 words) carry 3 marks, part D (Max. 400 words) carry 7 marks.

(iv) Except numericals, Derivation, Design and Drawing etc.

(a) Define DFA. List three household applications of finite automata. (See Unit-II, Page 85, Q.20)

(b) What is a trap state in FA ? State and explain the properties of transition functions. (See Unit-I, Page 4, Q.4)

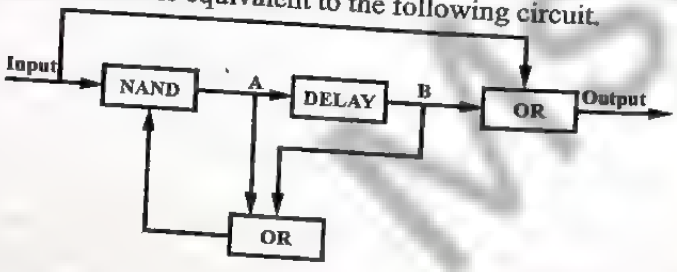
(c) Design deterministic fine automation accepting the following languages over the alphabet (0, 1) –

(i) The set of all words ending in 00.

(ii) The set of all words except $\in$.

(iii) The set of all words that begin with 0. (See Unit-I, Page 27, Prob.10)

(d) What do you mean by automata with output capability ? Draw a Mealy machine equivalent to the following circuit.




(See Unit-I, Page 15, Q.14)

Or

What do you mean by closure properties of regular languages ? State these properties. State pumping Lemma and show that $L = \{a^i b^i | i >= 1\}$ is not a regular language. (See Unit-II, Page 61, Q.18)





2. (a) Show that the following grammar is ambiguous
$S \rightarrow aSbS|bSaS|\in$ (See Unit-III, Page 110, Prob.11)

(b) What are leftmost and rightmost derivations ? Explain with suitable example. (See Unit-III, Page 97, Q.7)

(c) Why CFG is not considered adequate for describing natural language ? Explain with suitable example. (See Unit-III, Page 96, Q.2)

(d) What do you mean by normal forms ? Reduce the grammar G with following productions to CNF.
$S \rightarrow ASA|bA$
$A \rightarrow B|S$
$B \rightarrow c$

(See Unit-III, Page 122, Prob.31)

Or

What do you mean by useless production ? Consider the grammar $G = (V, T, P, S)$ where V, T, P, S are given as –
$V = \{S, A, B, C, E\}$
$T = \{a, b, c\}$
$S = \{S\}$ and
P consists of
$S \rightarrow AB$
$A \rightarrow a$
$B \rightarrow b$
$B \rightarrow C$
$E \rightarrow c$
Eliminate useless symbols and productions from the above grammar. (See Unit-III, Page 116, Prob.22)

3. (a) What is PDA ? Explain instantaneous description of PDA. (See Unit-IV, Page 156, Q.6)

(b) State the difference between PDA and the FA. (See Unit-IV, Page 156, Q.5)

(c) Design a PDA to accept he language $\{x \in (a, b)^* | n_a(x) > n_b(x)\}$. (See Unit-IV, Page 169, Prob.12)

(d) Consider the grammar –
$S \rightarrow aA$
$A \rightarrow aABC|bB|a$
$B \rightarrow b$
$C \rightarrow c$
Construct PDA corresponding to this grammar. Also provide moves of the PDA and the leftmost derivation for any string in the language defined by the grammar. (See Unit-IV, Page 191, Prob.32)

Or

The intersection of two context-free languages may or may not be context-free. (See Unit-III, Page 141, Q.24)

Also write an algorithm for a given any context-free grammar to determine whether or not it can generate any words.

4. (a) Differentiate the purpose of the study of Turing machine with Finite Automata/Pushdown Automata. (See Unit-V, Page 194, Q.3)

(b) What is Turing-computable function ? Define recursive function. (See Unit-V, Page 200, Q.9)

(c) How UTM overcomes the limitation of Turing machine ? Also define UTM. (See Unit-V, Page 205, Q.17)

(d) Present a Turing machine that inserts symbol # in the beginning of a string on the turing tape. Assume $\Sigma = \{a, b\}$. (See Unit-V, Page 227, Prob.14)

Or

Design a Turing machine that adds two numbers presented in binary notation and leaves the answer on the tape in binary form. (See Unit-V, Page 228, Prob.15)

5. (a) Define P and NP problems. (See Unit-V, Page 249, Q.49)

(b) Discuss tractable and intractable problem. (See Unit-V, Page 247, Q.43)

(c) Draw and explain commonly believed relationship between class P, NP, NP-complete and NP-hard. (See Unit-V, Page 251, Q.56)

(d) Define and discuss vertex cover problem. (See Unit-V, Page 256, Q.64)

Or

Discuss and explain travelling salesman problem. (See Unit-V, Page 264, Q.71)

## RGPV

**B.E. (Fifth Semester) EXAMINATION, June 2016**
**THEORY OF COMPUTATION**
**(CS-505)**

Note : (i) Answer five questions. In each question part A, B, C is compulsory and D part has internal choice.

(ii) All parts of each questions are to be attempted at one place.

(iii) All questions carry equal marks, out of which part A and B (Max. 50 words) carry 2 marks, part C (Max. 100 words) carry 3 marks, part D (Max. 400 words) carry 7 marks.

(iv) Except numericals, Derivation, Design and Drawing etc.

### Unit-I

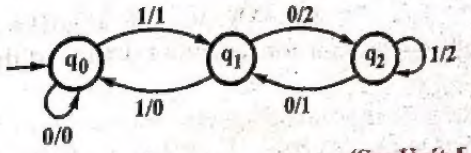
1. (a) Design DFA that accepts all strings with at most 3 a's. (See Unit-I, Page 22, Prob.2)

(b) Design a NFA for $\{cbab^n \mid n >= 0\}$. (See Unit-I, Page 22, Prob.4)

(c) Construct Moore machine for the following Mealy machine.








(See Unit-I, Page 44, Prob.34)

(d) Write and explain Myhill-Nerode theorem. (See Unit-II, Page 58, Q.14)

Or

Construct NFA for the following grammar –
$S \rightarrow Ab|ab, A \rightarrow Ab|Bb, B \rightarrow Ba|a$ (See Unit-I, Page 30, Prob.18)

### Unit-II

2. (a) Give CFG for R.E. $(011 + 1)^* (01)^*$. (See Unit-III, Page 153, Prob.65)

(b) Explain GNF conversion steps. (See Unit-III, Page 102, Q.15)

(c) Explain ambiguous grammar problem. (See Unit-III, Page 98, Q.9)

(d) Convert the following CFG to CNF

$S \rightarrow ASB \mid \in$
$A \rightarrow aAS \mid a$
$B \rightarrow SbS \mid A \mid bb$

(See Unit-III, Page 124, Prob.33)

Or

Convert the following grammar G into GNF

$S \rightarrow XA \mid BB$
$B \rightarrow b \mid SB$
$X \rightarrow b$
$A \rightarrow a$

(See Unit-III, Page 130, Prob.40)

### Unit-III

3. (a) Explain PDA. (See Unit-IV, Page 155, Q.2)

(b) Explain how many way's PDA can accept (final out null store). (See Unit-IV, Page 157, Q.7)

(c) Explain pumping lemma for CFL. (See Unit-III, Page 136, Q.18)

(d) Design pushdown automata which accepts $L = \{0^n 1^{2n} \mid n \geq 1\}$. (See Unit-IV, Page 164, Prob.3)

Or

Design a pushdown automata which accepts set of balanced parentheses. $(\{\{(\ )\}\})$ (See Unit-IV, Page 170, Prob.14)

### Unit-IV

4. (a) Explain ID of a turing machine. (See Unit-V, Page 199, Q.7)

(b) Explain multitape and universal turing machine. (See Unit-V, Page 206, Q.18)

(c) Explain Church's hypothesis. (See Unit-V, Page 212, Q.25)

(d) Design turing machine to add two numbers a and b. (See Unit-V, Page 235, Prob.22)

Or

Design turning machine for accepting strings of the language defined as $\{\omega\omega\gamma/\omega\in(0+1)^*\}$.

**Unit-V**

5. (a) Explain P and NP problems. (See Unit-V, Page 249, Q.49)
   (b) Difference between NP complete Vs NP hard problem. (See Unit-V, Page 249, Q.51)
   (c) Explain process of reducibility. (See Unit-V, Page 246, Q.42)
   (d) Describe hamiltonian path problem. (See Unit-V, Page 258, Q.67)

Or

Describe vertex cover problem. (See Unit-V, Page 258, Q.66)

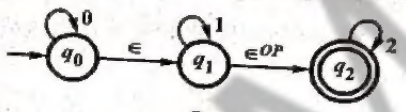
RGPV

**B.E. (Fifth Semester) EXAMINATION, Dec. 2016**
**THEORY OF COMPUTATION**
**(CS-505)**

Note : (i) Answer five questions. In each question part A, B, C is compulsory and D part has internal choice.
(ii) All parts of each questions are to be attempted at one place.
(iii) All questions carry equal marks, out of which part A and B (Max. 50 words) carry 2 marks, part C (Max. 100 words) carry 3 marks, part D (Max. 400 words) carry 7 marks.
(iv) Except numericals, Derivation, Design and Drawing etc.

**Unit-I**

(a) Define two way finite automata. (See Unit-I, Page 8, Q.9)
(b) Define regular expressions. (See Unit-II, Page 50, Q.6)
(c) Design DFA accepting the language over the alphabet 0, 1 that have the set of all strings ending in 00. (See Unit-I, Page 27, Prob.11)
(d) Determine the DFA equivalent to the given NFA.




Or (See Unit-I, Page 37, Prob.26)

Explain Myhill Nerode theorem with example. (See Unit-II, Page 58, Q.14)

**Unit-II**

(a) Define ambiguity. (See Unit-III, Page 98, Q.9)
(b) Define CFG. (See Unit-III, Page 95, Q.1)
(c) Find L(G) for the grammar $S \rightarrow aCa$, $C \rightarrow aCa/b$. (See Unit-III, Page 153, Prob.66)
(d) Prove that the given grammar is ambiguous.
(i) $S \rightarrow aSb/SS/\in$





(ii) $S \rightarrow AB/aaB$, $A \rightarrow a/Aa$, $B \rightarrow b$ (See Unit-III, Page 108, Prob.7)

Or

Eliminate unit, useless and $\in$-productions from the grammar $S \rightarrow aA/aBB$, $A \rightarrow aaA/\in$, $B \rightarrow bB/bbC$, $C \rightarrow B$. (See Unit-III, Page 118, Prob.25)

**Unit-III**

3. (a) Give the formal definition of PDA. (See Unit-IV, Page 155, Q.2)
   (b) Define deterministic PDA. (See Unit-IV, Page 160, Q.10)
   (c) Explain the pumping Lemma of context free languages. (See Unit-III, Page 136, Q.18)
   (d) Design PDA to accept the language $L(G) = \{a^n b^m a^n/m, n\geq 1\}$. (See Unit-IV, Page 183, Prob.20)

Or

Design PDA to accept the language $L(G) = \{ww^R/w\in(0,1)^*$ and $w^R$ is the reverse of word w$\}$. (See Unit-IV, Page 163, Prob.2)

**Unit-IV**

4. (a) What are decidable and undecidable problems ? (See Unit-V, Page 246, Q.40)
   (b) Give two properties of recursively enumerable set. (See Unit-V, Page 240, Q.29)
   (c) Explain the types of turing machine. (See Unit-V, Page 209, Q.23)
   (d) Design turing machine for the language $L(G) = \{a^n b^n/n\geq 1\}$ (See Unit-V, Page 225, Prob.13)

Or

Design turing machine for the language $L(G) = \{a^n b^m a^{n-m}/n\geq 1, m\geq 1\}$. (See Unit-V, Page 224, Prob.12)

**Unit-V**

5. (a) What is NP hard problems ? (See Unit-V, Page 251, Q.57)
   (b) How P class problems different from NP class problem ? (See Unit-V, Page 248, Q.48)
   (c) What is hamiltonian path problem ? Explain. (See Unit-V, Page 258, Q.67)
   (d) Differentiate between tractable and untractable problems. (See Unit-V, Page 247, Q.43)

Or

Explain vertex cover problem briefly. (See Unit-V, Page 258, Q.66)

RGPV

**IT-5001 (CBGS)**
**B.E. V Semester**
**EXAMINATION, Dec. 2017**
**Choice Based Grading System (CBGS)**
**THEORY OF COMPUTATION**

Note : (i) Attempt any five questions.
(ii) All questions carry equal marks.

1. (a) Construct a DFA equivalent to an NDFA whose transition table is defined by – 7

| State | a | b |
|---|---|---|
| $q_0$ | $q_1, q_3$ | $q_2, q_3$ |
| $q_1$ | $q_1$ | $q_3$ |
| $q_2$ | $q_3$ | $q_2$ |
| $q_3$ | – | – |

(See Unit-I, Page 33, Prob.21)

(b) Construct a DFA accepting all strings w over {0, 1} such that the number of 1's in w is 3 mod 4. (See Unit-I, Page 30, Prob.17) 7

2. (a) Prove $(1 + 00^*1) + (1 + 00^*1)(0 + 10^*1)^*(0 + 10^*1) = 0^*1(0 + 10^*1)^*$. (See Unit-II, Page 70, Prob.13) 7

(b) Write the identities of regular expression. 7

(See Unit-II, Page 51, Q.7)

3. Construct a context free grammars to generate the following – 14

(i) $0^m1^m$ $m \geq 0$
(ii) $0^m1^n$ $1 \leq m \leq n$
(iii) $0^m1^n2^r$ $m = n$
(iv) $0^l1^m2^r$ $l + m = n$

(See Unit-III, Page 150, Prob.61)

4. Construct a minimum state automation equivalent to a given automation M whose transition table is defined below – 14

| State | Input | |
|---|---|---|
| | a | b |
| $\rightarrow q_0$ | $q_0$ | $q_3$ |
| $q_1$ | $q_2$ | $q_5$ |
| $q_2$ | $q_3$ | $q_4$ |
| $q_3$ | $q_0$ | $q_5$ |
| $q_4$ | $q_0$ | $q_6$ |
| $q_5$ | $q_1$ | $q_4$ |
| $(q_6)$ | $q_1$ | $q_3$ |

(See Unit-II, Page 93, Prob.39)

(a) Let G be the grammar
$S \rightarrow OB|1A$, $A \rightarrow O|OS|1AA$, $B \rightarrow 1|1S|OBB$
For the string 00110101 find – 7
(i) LMD
(ii) RMD
(iii) Parse tree.

(See Unit-III, Page 106, Prob.3)





(b) If G is the grammar $S \rightarrow SbS|a$, show that G is ambiguous. 7

(See Unit-III, Page 109, Prob.8)

6. (a) Define a PDA ? Construct a PDA equivalent to the following context free grammar $S \rightarrow OBB$, $B \rightarrow OS|1S|O$. Test whether $010^4$ is in N(A). (See Unit-IV, Page 187, Prob.27) 7

(b) Construct a PDA for the following language – 7
(i) $a^nb^n$ $n \geq 0$
(ii) $a^nb^{2n}$ $n \geq 1$

(See Unit-IV, Page 166, Prob.9)

7. (a) Explain P and NP type of problem. Write any three examples of P or NP type problem. (See Unit-V, Page 248, Q.47) 7

(b) Describe decidable and undecidable problem. Explain Halting problem. (See Unit-V, Page 246, Q.41) 7

8. Write a short notes (any three) – 14
(a) Turing machine (See Unit-V, Page 206, Q.19)
(b) Universal turing machine (See Unit-V, Page 206, Q.19)
(c) NPDA and DPDA (See Unit-IV, Page 160, Q.12)
(d) Closure property of regular grammar. (See Unit-II, Page 55, Q.12)

RGPV

**IT-5001 (CBGS)**
**B.E. V Semester**
**EXAMINATION, November 2018**
**Choice Based Grading System (CBGS)**
**THEORY OF COMPUTATION**

Note : (i) Attempt any five questions.
(ii) All questions carry equal marks.

1. (a) Design finite automata for the given regular expression $(a + b)^*abb$. (See Unit-II, Page 70, Prob.12)

(b) Differentiate between Mealy and Moore machine. (See Unit-I, Page 13, Q.13)

2. (a) Mention the closure properties of regular languages. (See Unit-II, Page 55, Q.12)

(b) Explain Myhill-Nerode method of minimization. (See Unit-II, Page 60, Q.16)

3. (a) State and prove the pumping lemma theory of regular language. (See Unit-II, Page 60, Q.17)

(b) Define two way finite automata with suitable example. (See Unit-I, Page 8, Q.9)

4. (a) Define leftmost derivation, rightmost derivation and parse tree with suitable example. (See Unit-III, Page 98, Q.8)

(b) Convert the following grammar into GNF –

$S \rightarrow AA/0$

$A \rightarrow SS/1.$

(See Unit-III, Page 129, Prob.39)

5. (a) Explain Chomsky classification of languages with suitable example.

(See Unit-III, Page 96, Q.3)

(b) What is ambiguity in grammar ? Show that the given grammar is ambiguous.

$E \rightarrow E + E / E*E / 2 / 3 / 4.$

(See Unit-III, Page 110, Prob.12)

6. (a) Design PDA to accept $\{WW^R / W \in \{0,1\}^*\}$, where W is word and $W^R$ is reverse of word.

(See Unit-IV, Page 163, Prob.2)

(b) Design PDA corresponding to given CFG

$S \rightarrow aSA$

$S \rightarrow bSb$

$S \rightarrow c$

(See Unit-IV, Page 187, Prob.26)

7. (a) Construct a turing machine that accepts the language $L = \{a^n b^{2n}\}$.

(See Unit-V, Page 230, Prob.17)

(b) What do you mean by turing machine ? Explain multiple tapes turing machine.

(See Unit-V, Page 204, Q.13)

8. Write short notes (any three) –

(a) Recursive and recursively enumerable language

(b) Halting problem

(c) NP complete Vs NP hard

(d) CNF

(f) NPDA.

(See Unit-V, Page 252, Q.58)

OOO